International Journal of Computer Engineering & Technology (IJCET)
Volume 10, Issue 2, March-April 2019, pp. 215-233, Article ID: IJCET_10_02_022
Available online at
http://iaeme.com/Home/issue/IJCET?Volume=10&Issue=2
Journal Impact Factor (2016): 9.3590(Calculated by GISI) www.jifactor.com
ISSN Print: 0976-6367 and ISSN Online: 0976–6375

DISTRIBUTED CONTROLLER FAULT TOLERANCE MODEL (DCFT) USING LOAD BALANCING IN SOFTWARE DEFINED NETWORKING

Gaurang Lakhani and Dr. Amit Kothari
Research Scholar Ph.D. (CE/IT) Gujarat Technological University,
Ahmedabad, Gujarat, India

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, load collection, 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 design 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. We compare packet delay, communication overhead and load balancing rate in a custom topology 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 Tolerance, DCFT, Switch Migration, coordinator Election, Load Balancing, Data Plane, and Control Plane.

Cite this Article: Gaurang Lakhani, Dr. Amit Kothari, Distributed Controller Fault Tolerance Model (DCFT) Using Load Balancing in Software Defined Networking, *International Journal of Computer Engineering and Technology,* 10(2), 2019, pp. 215-233.
http://iaeme.com/Home/issue/IJCET?Volume=10&Issue=2

1. INTRODUCTION

Software-Defined Networking (SDN) is a new approach in 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. The brain of the control plane is the SDN controller. Controller talks with network devices through southbound Interface (SBI) such as openflow protocol. The control plane exposes some features and APIs through the Northbound Interfaces (NBI) to network operators to design various management application exploiting such as a set of REST API. East-West bound API used for inter-controller communication among multiple controllers. Control functionality is removed from network devices that are considered as simple packet forwarding elements. The forwarding decision is flow based rather than destination based. Figure 1 shows flow tables, flow defined by a set of packet field values acting as match criteria and sets of associated actions to the matching criteria. All packets of the same flow receive identical service policies at the forwarding devices. The flow abstraction allows unifying the behavior of different types of network devices including routers, switches, firewalls, and middleboxes. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables. The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between switches and SDN controller. Openflow switch has one or more tables of packet handling rules. Each rule matches a subset of traffic and performs certain actions on traffic. Depending on the rules installed by controller application, an OpenFlow switch instructed by controller to behave like a router, switch, firewall or perform any other roles.

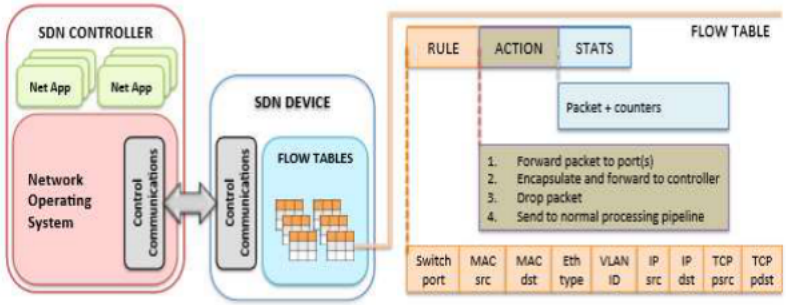

Figure 1 Openflow enabled SDN device

SDN adoption raises issues of scalability and reliability in centralized design. That can be referenced with physical delegation of the control plane. Such materially dispersed but logically centralized systems bring an added set of challenges. Logically centralized controllers responsible for forwarding routing decisions, controller failure is a significant problem of SDN, a load of the futile controller has to be spread among the other controllers. In this paper, we design a Distributed Controller Fault Tolerance model (DCFT) using load balancing in SDN.

Few previous papers [5][6][7][8][9] have explored switch migration to provide load balancing but existing proposed algorithms can only work with load imbalance they cannot work with the event of controller failure. Control plane is suffering from a lack of fault tolerance. For a distributed control plane coordinator election algorithm used to identify unique coordination between all SDN controllers.

The main contribution of this paper is 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 design could achieve load balancing among distributed controllers while fault occurs, regardless of network traffic variation and outperform static binding controller system with communication overhead, controller load balance rate, and packet delay. Verify the DCFT model on custom topology.

Rest of the paper is organized as follows. Section II presents literature survey Section III presents 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.

2. LITERATURE SURVEY

2.1. Distributed control plane in SDN

Distributed SDN control plane architecture 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 subset of SDN switches. ONOS [15], Onix [22], HyperFlow [23] and OpenDayLight [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 responsibility 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 build global network view and each controller assumes responsibility for its system. The synchronization between controllers ought to be declared for certain occasion such as link status changes that could affect the network view. The OpenDayLight [24] controller starts by building the data structure trees by utilizing the Yang modeling language and MD-SAL. They concentrated on necessary components to achieve spread control plane, give a worldwide perspective of the network topology for upper 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 assumes that the network control plane is vertically divided into a different dimension (layers) reliant upon the required 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 proposes to diminish the general weight of the control plane without the need to alter openflow switches. It set up two dimensions control plane where frequent events occurring near the data path are handled by the bottom layer, and non-local events requiring network vide view dealt by the top layer.

2.2. 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 allows a set of nodes or servers to work together as a unique coherent system that is able to handle failures of some of its nodes. It can be done by replicating state machine of the coordinator

2.3. Switch migration algorithm.

Dixit et al. [5] work towards the utilization of controller resources using load balancing and reduce the power consumption by switching of under loaded controllers from the controller pool. Dixit, Advait Abhay, et al. [6] 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 calculate the load and take 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. Centralized node used for load balancing, a centralized controller is constrained by memory, CPU power, and bandwidth. Moreover, a centralized node collects load information intermittently and it talks a lot of messages frequently with other controllers, which will prompt to performance reduction of the whole system. Aly et al. [18] mentioned the selection of destination backup controller based on a distance between switches and target controller,

current load and percentage of packet loss. Distance between a switch and backup controller influence the packet response time. Which affects 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 messages transactionally and precisely once (at both the controllers and the switches). Ravana keeps up these certifications even with both 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.

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 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. unlike the distributed approach where the controller will need to collect information from each switch. Switch loss connection with a controller checked by probe message sent periodically to the controller.

Hu tao et al [31] depicted distributed decision mechanism (DDM) based 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 according to the selection probability, and the target controllers are dictated by integrating three network costs, including information accumulation, switch migration, and arranged switch movement. Finally, 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.

3. PROPOSED SYSTEM ARCHITECTURE OF THE DCFT MODEL

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

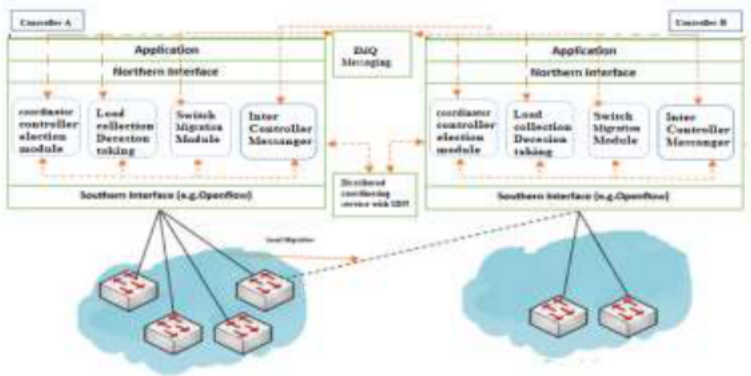

Figure 2 Architecture of the distributed control plane with load balancing [10]

(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 periodically receives the current load of each controller and switches load. The controller's current load is the value that specifies the load of the controller at a given time. The load represents a number of flows per second that the controller receives from the switches. Coordinator controller checks periodically status of the controllers. To detect the failure of the controller, coordinator controller uses controller information. For every specific time coordinator controller checks the last updated time of the controller's current load. If the last updated time exceeds a certain threshold, the coordinator controller considers this controller as a failed controller and takes the next step to recover the controller failure.

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 status of the controllers.

(c) Load Calculation and decision taking module

In 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 very huge flow table and a load of the controller will be high. Bigger average message arrival rate of a switch means this switch conveys more load to the controller. Propagation delay also impact factor. If the controller is overloaded, we choose to switch to migrate according to 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 based 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. After a specified time interval of every 5 seconds, the load calculation module calculates the load and send 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

CurrentLoad= Controller's Current Load

PreviousLoad= 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 a 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 selection of destination backup controller based on distance between switches and target controller, current load and percentage of packet loss. Distance between 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 nearest backup controller with considering outstanding workload on the destination backup controller steps of assignment of the switch as follows.

1. Assign each switch to n backup destination controller can be from sorted array list of the closest controller. array list stored at the distributed data store.
2. Each timespan 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. After the coordinator controller detect which controller failed, coordinator controller detects the switches of the failed controller.
6. Loop through the failed controller's associated changes of switches to check the backup controllers list.
7. Check the availability of each backup controller in the backup controller list.

8. In the event of first backup 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 checks the next available backup controller.

10. Steps 2 to 9 repeated until coordinator controller allots switch to a suitable backup controller while the controller load changes over time.

4. 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 switch controller mapping 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.

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.

(a) 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 failed controller to a lightest loaded controller by proposed switch migration. All the controllers may have a different number of switches. Figure 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.

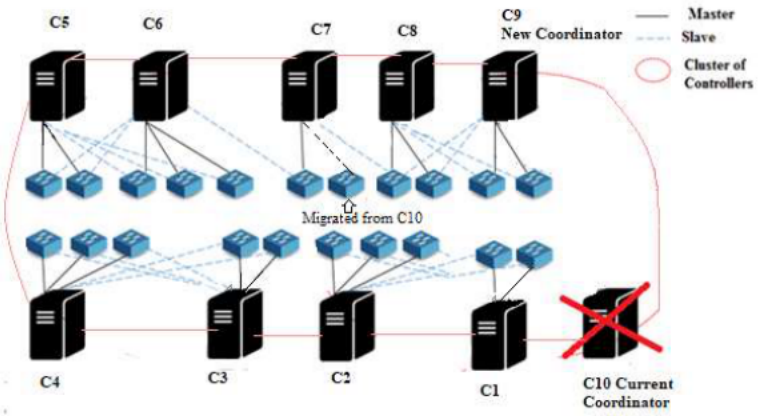

Figure 3 Failure in coordinator controller, election of new coordinator controller

(b) 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.

(c) 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.

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 role 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 role.

The default job of a controller is OFPCR_ROLE_EQUAL [19]. In this job, the controller has full access to the switch and is equal to other controllers in a similar job. As a matter of the course, the controller gets all the switch nonconcurrent messages (such as packet-in, flow-removed). The controller can send controller-to-switch directions to alter the conditions of the switch. The switch does not do any intervention or asset sharing between controllers.

A controller can demand its job to be changed to 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-status 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 access 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 role to OFPCR_ROLE_MASTER, the switch changes 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 performs such job changes, if a controller job is changed 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 time controller is never again reachable, and the switch will most likely to transmit that occasion).

Each controller may send 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 change 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 changed 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 everyone 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.

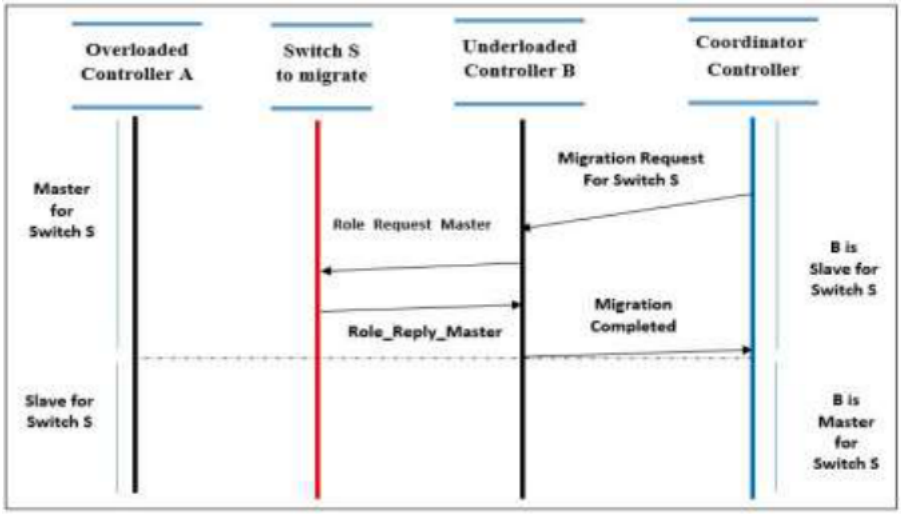

Figure 4 Proposed switch migration process in overloading of the controller

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.
- After the receipt of the load migration request selected destination controller send role change request (from slave to master) to the switch which needs to be migrated.
- 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.
- Destination controller sends End Migration message to the Coordinator. Coordinator update controller switch mapping in a Distributed Database.

5. SIMULATION ANALYSIS

We use experimental testbed for simulation as mentioned in table1. Physical devices contain four machines with the configuration mentioned in table1. In the cluster, there is only one

master controller, which enables programmed network management. We design a series of experiment 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 closest underloaded controller to solve the load imbalance problem in CRD. In MUSM overloaded controller migrates switch into the controller that has a maximum residual 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. A slave controller will be chosen as a new master if the original master fails. We consider custom topology in figure 5. Traffic patterns are shown in table2 used for all simulations.

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 distributing the controller's load among other controllers whenever a point of disappointment occurs. 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 Number	Traffic source	Traffic destination
T1	H1	H4
T2	H8	H12
T3	H13	H18

We use hping3 to generate TCP flows to simulate the distribution 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 reduce 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.

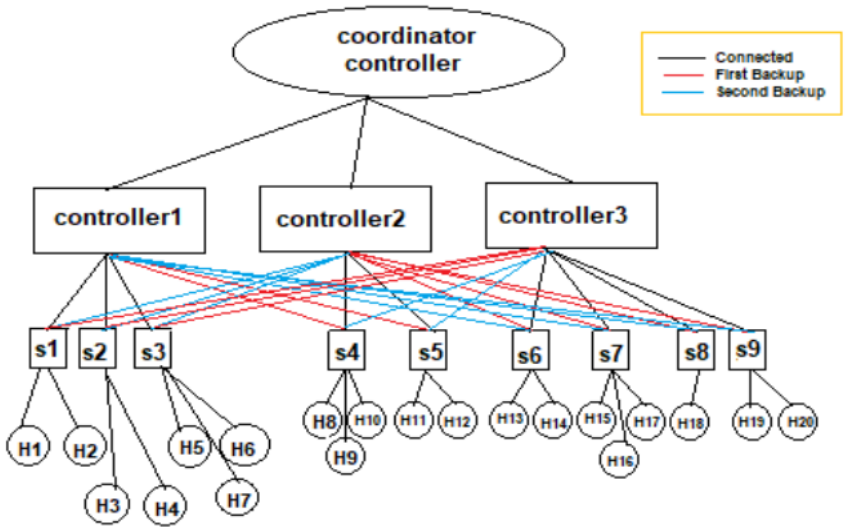

Figure 5. The logical perspective of the topology used in a simulation

Consider the topology shown in figure 5. DCFT model takes delay between switches and their associated controllers to minimize the response time. Table 3 shows a configuration of topology before switch migration while table 4 shows a configuration of topology after migration of switches from controller C2 to C1.

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 15 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 C3. 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 distributing the load of the failed controller C2 among C1 and C3. This migration causes an expanded number of solicitations to every controller then the blockage in this controller lead to the packet delay. The maximum packet delay for traffic T1 is 28.66 ms at 57 seconds, for traffic T2 is 27.99 ms at 59 seconds and for traffic, T3 is 33.6 ms at 55 seconds. Numerical results shown in figure 10 depicts the lowest packet delay by DCFT model compared to other methods of switch migrations. Packet delay reduced by our model is 28.52 %. 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 count of the given topology shown in figure 6 When the load imbalance occurs, packet delay (latency) increased, we change flow request count to overload controller and observe packet delay of the given topology in figure 9.

Communication overhead

The Communication overhead is created between switch-controller and between controller-controller. Rule installation in openflow switches causes wasteful network operation inferable from the high overhead potential on the openflow controller. DCFT model demonstrated pursued routine with regards to install a rule in the switches for a minimal flow entry in the

network switches without impairing the network operation itself. Figure 8 depicts a communication overhead for the given topology.

Since ZSM just arranges single controller, the correspondence overhead between 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 selection of target controller, which brings down the overhead between controllers. On the other hand, closest migration is easy to produce traffic congestion that may increase communication overheads between switches and controllers. if multiple switches swarm into nearest controller at the same time. MUSM lessens overhead by adding an extra 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 match actions information so that the subsequent 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%

Controller load balancing rate

We record the number of requests processed by each controller and reflect the distribution of controller loads. ZSM has only one controller It doesn't exist load balancing. We compare the result of CRD, MUSM, and DCFT for three controllers in the given topology. which are shown in figure 9. CRD has a big difference in the number of requests processed by each controller. MUSM on second place and DCFT has the slight fluctuation. As CRD migrates the switch to the closest controller, the neighbors of the overloaded controller are likely to produce switch migration again if receiving too many migrating switches. Controller load balancing rate increased by 7.11 %

Table 3 Before Switch Migration

Controller	Type Master/Slave	Switch	Type of Switch	No of the host under the switch
Coordinator Controller	Master	S1,S4,S6 attached via C1,C2,C3	Openflow	
C1	Master	S1	Openflow	H1,H2
		S2	Openflow	H3, H4
		S3	Legacy	H5,H6,H7
C2	Slave	S4	Openflow	H8,H9,H10
		S5	Openflow	H11,H12
C3	Equal	S6	Legacy	H13,H14
		S7	Openflow	H15,H16,H17
		S8	Openflow	H18
		S9	Openflow	H19, H20

Table 4 After switch Migration from C2 to C1(S4, S5 migrated to C1)

Controller	Type Master/Slave	Switch	Type of Switch	No of the host under the switch
Coordinator Controller	Master	S1,S4,S6 attached via C1,C2,C3	Openflow	
C1	Master	S1	Openflow	H1,H2
		S2	Openflow	H3, H4
		S3	Legacy	H5,H6,H7
		S4	Openflow	H8,H9,H10
		S5	Openflow	H11,H12
C2(crashed after 15s)	Slave	S4	Openflow	H8,H9,H10
		S5	Openflow	H11,H12
C3	Equal	S6	Legacy	H13,H14
		S7	Openflow	H15,H16,H17
		S8	Openflow	H18
		S9	Openflow	H19, H20

Table 5 Maximum packet delay(ms),Communication overhead and load balancing rate (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		Switch-controller	Controller-controller	C1	C2	C3
T1	40.1	423	372	503	511	475	28.66	237	193	496	521	489
T2	30.9	394	321	478	524	434	27.99	206	134	492	536	442
T3	16.8	254	212	474	504	468	17.2	148	126	483	514	478

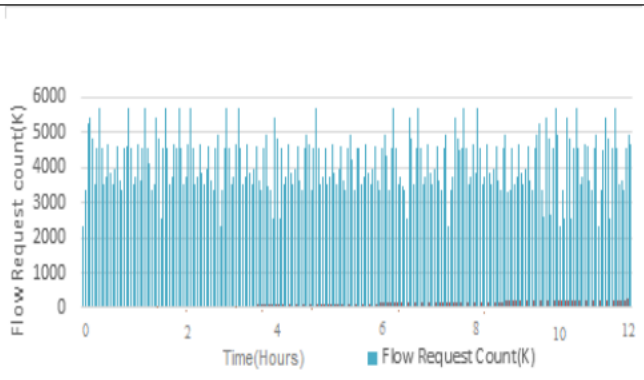

Figure 6 Network traffic in custom topology

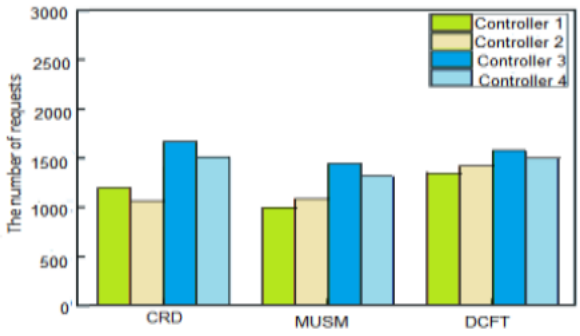

Figure 7 Request processed by each controller-custom topology

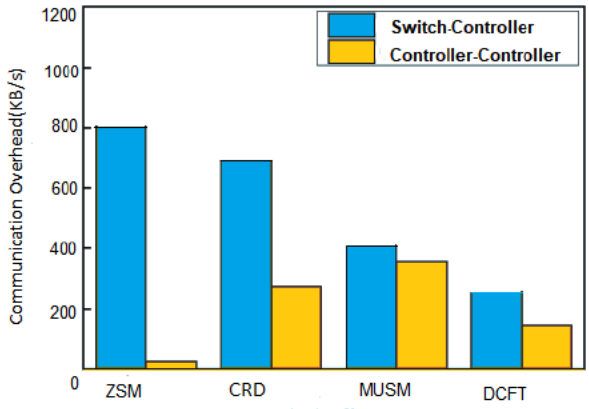

Figure 8 Communication Overhead in custom topology

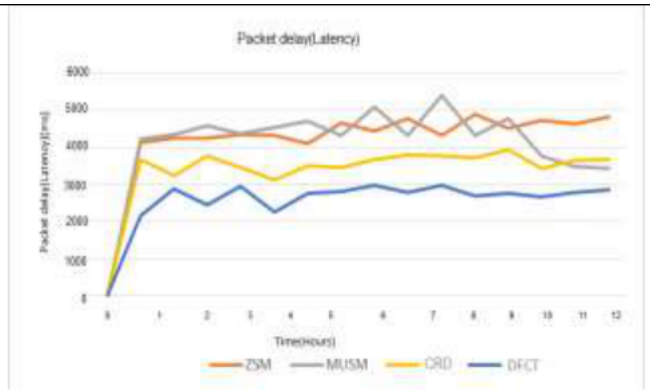

Figure 9 Packet delay(latency) in custom topology

6. 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 flat SDN controller and hierarchical SDN controller discussed.

Simulation analysis performed with series of experiments performed using traffic patterns (Table 2), on custom topology with three controllers along with coordinator controller Communication overhead, controller load balance rate, packet delay used as evaluation indexes. It is found in figure 7 that DCFT model reduces packet delay by 24.51 %. From figure 8 Average communication overhead is reduced switch-controller by 44.47 % and controller-controller is by 48.12%. From figure 9 Controller load balancing rate is increased by 7.11 %. 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 future work focuses on an analysis of the failure of both switch and controller and finds a more refined technique to distribute a load of futile controller among other controllers founded on AI-based techniques.

REFERENCES

[1] Yu, Yinbo, et al. "Fault Management in Software-Defined Networking: A Survey." IEEE Communications Surveys & Tutorials (2018).

[2] P. Peresini, M. Ku´zniar, and D. Kosti´c, “Monocle: Dynamic, fine grained data plane monitoring,” in Proc. of ACM CoNEXT, 2015.

[3] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock et al., “Troubleshooting blackbox SDN control software with minimal causal sequences,” in Proc. of ACM SIGCOMM, 2014, pp. 395–406.

[4] K. Mahajan, R. Poddar, M. Dhawan, and V. Mann, “JURY: Validating Controller Actions in Software-Defined Networks,” in 46th IEEE/IFIP DSN, 2016, pp. 109–120.

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

[6] Dixit, Advait, Fang Hao, Sarit Mukherjee, T. V. Lakshman, and Ramana Rao Kompella. "ElastiCon; an elastic distributed SDN controller." In *Architectures for Networking and*

Communications Systems (ANCS), 2014 ACM/IEEE Symposium on, pp. 17-27. IEEE, 2014.

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

[8] Chen, Yanyu, Qing Li, Yuan Yang, Qi Li, Yong Jiang, and Xi Xiao. "Towards adaptive elastic distributed Software Defined Networking." In *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*, pp. 1-8. IEEE, 2015.

[9] 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.

[10] 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.

[11] Yu, Jinke, Ying Wang, Keke Pei, Shujuan Zhang, and Jiacong Li. "A load balancing mechanism for multiple SDN controllers based on load informing strategy." In *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*, pp. 1-4. IEEE, 2016.

[12] A.Dixit,F.Hao,S.Mukherjee,T.V.Lakshman. ElastiCon: an elastic distributed SDN controller. Proceedings of 2014ACM/IEEE Symposium on architectures for Networking and Communications Systems (ANCS), Marina, 2014:17-27.

[13] Guozhen Cheng, Hong chang Chen, Zhiming Wang. DHA:Distributed decisions on the switch migration toward a Scalable SDN control plane.Proceedings of 2015 IFIP Networking Conference,Toulouse,2015:1-9.

[14] Jarraya, Yosr, Taous Madi, and Mourad Debbabi. "A survey and a layered taxonomy of software-defined networking." *IEEE communications surveys & tutorials* 16, no. 4,2014.

[15] Berde, Pankaj, et al. "ONOS: towards an open, distributed SDN OS." Proceedings of the third workshop on hot topics in software defined networking. ACM, 2014

[16] Mantas, André Alexandre Lourenço. Consistent and fault-tolerant SDN controller. Diss. 2016.

[17] Hunt P, Konar M, Junqueira F P, et al., Zookeeper: wait-free coordination for internet-scale systems[c], Proceedings of the 2010 USENIX conference on USENIX annual technical conference. 2010, 8:11-11

[18] Aly, Wael Hosny Fouad, and Abeer Mohammad Ali Al-anazi. "Enhanced Controller Fault Tolerant (ECFT) model for Software Defined Networking." Software Defined Systems (SDS), 2018 Fifth International Conference on. IEEE, 2018

[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, Yannan, et al. "Balanceflow: controller load balancing for openflow networks." Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on. Vol. 2. IEEE, 2012

[22] Koponen, Teemu, et al. "Onix: A distributed control platform for large-scale production networks." OSDI. Vol. 10. 2010

[23] Tootoonchian, Amin, and Yashar Ganjali. "Hyperflow: A distributed control plane for openflow." Proceedings of the 2010 internet network management conference on Research on enterprise networking. 2010.

[24] Medved, Jan, et al. "Opendaylight: Towards a model-driven sdn controller architecture." 2014 IEEE 15th International Symposium on. IEEE, 2014

[25] Aly, wael Hosny Fouad. "A Novel Fault Tolerance Mechanism for Software Defined Networking." 2017 European Modelling Symposium (EMS). IEEE, 2017

[26] Estban Hernandaz "Implementation and performance of a SDN cluster-controller based on the OpenDayLight framework." Ph.D. thesis (2016).

[27] Katta, Naga, et al. "Ravana: Controller fault-tolerance in software-defined networking." Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research. ACM, 2015.

[28] Botelho, Fábio, et al. "On the design of practical fault-tolerant SDN controllers." Software Defined Networks (EWSDN), 2014 Third European Workshop on. IEEE, 2014.

[29] Obadia, Mathis, et al. "Failover mechanisms for distributed SDN controllers." Network of the Future (NOF), 2014 International Conference and Workshop on the. IEEE, 2014

[30] Fonseca, Paulo, et al. "Resilience of sdns based on active and passive replication mechanisms." Global Communications Conference (GLOBECOM), 2013 IEEE. IEEE, 2013

[31] Hu, Tao, et al. "A distributed decision mechanism for controller load balancing based on switch migration in SDN." China Communications 15.10 (2018): 129-142

[32] Hassas Yeganeh, Soheil, and Yashar Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[33] curtis, Andrew R., et al. "DevoFlow: Scaling flow management for high-performance networks." ACM SIGCOMM Computer Communication Review. Vol. 41. No. 4. ACM, 2011

[34] Yu, Minlan, et al. "Scalable flow-based networking with DIFANE." ACM SIGCOMM Computer Communication Review41.4 (2011): 351-362.

[35] Lantz, Bob, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks." Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010

[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, Wael Hosny Fouad. "LBFTFB fault tolerance mechanism for software defined networking." Electrical and Computing Technologies and Applications (ICECTA), 2017 International Conference on. IEEE, 2017.

Mr. Gaurang Lakhani has completed his M.E. (CSE) from Gujarat Technological University Ahmedabad in 2013. And joined PhD from 2014 in same university. His main area of research in Distributed SDN. He had published 5 National/International research papers. Mainly Interested computer Networks, Network/ Information security, Distributed computing, Worked as reviewer /session chair in National level Conferences. Worked as committee members in GTU academic activities.

Dr. Amit D Kothari has completed his PhD from Hemchandracharya North Gujarat University, Patan, Gujarat in 2011. During PhD his research area was routing protocols for Mobile Adhoc Network (MANet). 6 students are pursuing their PhD research work under him. More than 13 National/International research paper are published. Delivered expert talk at many institutes and universities during his 15 years of academic career. His interest area is – Computer Network, Network / Information security, Parallel / Distributed Computing and Artificial Intelligence. He offers his service as a book reviewer for McGraw Hill Publication. He is invited as a member Doctorate Review Committee at many Universities