Fault management through load balancing in Distributed controller with SDN

Ph.D. Synopsis

For the Degree of Doctor of Philosophy
In Computer / IT Engineering

Submitted By

Lakhani Gaurang Vinodray

(Enrollment No: 149997107005, Batch: 2014)

(gvlakhani1@gmail.com)

Supervisor

Dr. Amit Kothari,
(amitdkothari@gmail.com)
Sr. Software engineer,
Accenture, Pune, India

Co-Supervisor

Dr. Subra Ganesan
(ganesan@oakland.edu)
Professor, Electrical and Comp Engg,
Oakland University, Rochester, USA

DPC Member 1

Dr. Bhushan Trivedi,
(bhtrivedi@gmail.com)
Director,
GLS Institute of Computer Technology,
Ahmedabad, India

DPC Member 2

Dr. Satyen Parikh
(parikhsatyen@gmail.com)
Dean, Faculty of Computer applications
Ganpat University, Kherva,
Maheana, India

Submitted to

Gujarat Technological University

Table of Contents

1. Abstract

Tremendous amount of data generated due to increasing no of users every day in the technology world. It is difficult to manage huge amount of discrete data with the traditional network. Even it is difficult to manage with technologies such as big data, cloud computing in traditional network. Software defined networking brings all the functionalities to a single location and making centralized decision. Controllers are the main entities of the SDN design, which governs decisions while routing packets. Centralized decision increases performance of the network. Distributed SDN controllers are physically distributed and logically centralized.

In this research work, we presented basics of Distributed SDN controllers with its properties. It also presents design choice of distributed SDN controllers and its analysis based on switch to controller connection, network information distribution strategy, controller coordination strategy, and In-band vs. Out-of-band connection strategy. We studied classification of existing SDN controllers vide their logical design (hierarchical/flat model), programming language, modularity, application domain, etc. We categorize fault management issues in management plane, control plane and infrastructure plane and their interface. Extensive literature survey ends with the research gap and eventually, concentrated on the faults generated through overloading of the controllers. Our research work intended to develop a model of robust distributed SDN controller. This novel model named as a DCFT (Distributed Controller Fault Tolerance) model. The model added with additional fault tolerance plane between application plane and control plane in the SDN stack. Model contains modules at different planes in the SDN stack. Data plane contains switches, controllers are managed at control plane. Internal communication among the controllers are managed with internal communication module at control plane. Load calculation and decision-making module helps in calculation of the load of each SDN controller and forward it to coordinator controller election module. It will decide coordinator controller of the model. Switch migration module at the fault tolerance plane migrates the switches to the least loaded controller on overloading. DCFT module manages the state of all the controllers. For the sample, three fault cases are managed by the fault tolerance module. Transaction module provides an interface called by the SDN application to achieve ACID properties. It will take SDN

update from many applications, devises the ACID execution of the concurrent updates, and at the end binds the updates to the network. We evaluate performance of the model (1) with and without coordinator controller (2) before and after switch migration and (3) in transaction management module. We concluded that model reduces in packet delay, rate of load balancing, and eventually achieve higher throughput at the cost of communication overhead.

Our model provides strongly consistent, fault tolerance control plane in distributed SDN controller and behave the same as fault free distributed SDN for its users (end-hosts and network application).

2. Brief description of the state of the art of the Research topic

Distributed controllers with SDN are physically distributed, logically centralized. It generates issues like scalability, reliability, availability, consistency, security, etc. There are few papers ONOS [1], ONIX [2], Hyperflow [3] and OpenDayLight [4] which implements distributed SDN controllers. They have focused on the essential components to accomplish a distributed control plane and provide a worldwide view of the network topology for the upper application. We can improve scalability, availability with the help of multiple controllers, but how to maintain synchronization between the switch and controller in case of one of the controllers is overloaded? Even careful deployment of switches and controllers does not guarantee for controllers to adjust changeful traffic load.

Actual networks show the appearances from two aspects: temporal and spatial [5]. For the moment, from the temporal angle, there may be a smaller amount of traffic during the night. However, there may be a huge scale of flows generated by the application of routing calculation in a very short time. Also, from this point of- a spatial, application running on different controllers possibly compute and generate a different number of flows, and some switches can get lots of flows compared to other domain of the network. So it is crucial for us to balance the load dynamically among the distributed controller cluster instead of a static network configuration. Overloaded controllers should be noticed and highly loaded switches mapped to this controller ought to be smoothly migrated to the underloaded controllers so as to improve resource utilization of the controller cluster.

A fault may occur at the link, switch, or controller level due to their failures in distributed SDN controller. Whole system or part of the system will be in halt state while fault occurred. The effective fault-tolerant solution needed in distributed SDN controller. **Our endeavor concluded in the following problem statement of the research.**

3. Definition of the problem

To propose a model for providing fault tolerance in the distributed SDN controller, as a side effect, our model is also able to achieve reduction in a few critical parameters including packet delay, rate of load balancing, and eventually able to achieve higher throughput at the cost of communication overhead.

4. Research Objective and Scope of work

From the reviewed literature following objectives have been derived:

- For designing robust distributed SDN controller, proposing switch migration algorithm and transaction management technique among controllers. Incorporating this technique proposing a novel, flat, Master/Slave connection, Coordinator based, load balancing model which helps to achieve better throughput, fault tolerance, reduction in packet delay, packet loss at the cost of reasonable communication overhead.

- Introduced a coordinator controller election algorithm which makes distributed SDN more reliable, consistent, and scalable. Evaluate the performance of the controllers (with and without coordinator controller).

- The proposed model includes transaction management module for providing update service in the SDN applications and transactionally update the network. SDN updates devises the ACID execution provides consistency guarantee with respect to packet loss, failure recovery.

- The proposed model includes fault tolerance and transaction management modules to provide strongly consistent, fault tolerant control plane for distributed SDN controller and behave the same as a fault-free distributed SDN for its users (end - hosts and network applications).

- Evaluate the performance of the controller (before and after switch migration) for the proposed model with reference to packet delay, rate of load balancing, eventually achieve higher throughput at the cost of communication overhead.

5. Original contribution by the thesis

- The thesis proposed a Distributed Controller Fault Tolerance (DCFT) model which contains data plane, control plane, and application (management) plane including one additional sub layer called fault tolerance plane in SDN stack, which is extended from application plane. Fault tolerance sub layer comprises different modules such as switch migration module, DCFT module, fault tolerance module, and transaction management module.
- 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, rate of load balancing, and packet delay and throughput.
- Verify the DCFT model on custom topology and other well-known topologies.
- Derived and compare results of before and after switch migration, with and without coordinator controller, and overhead management in transaction management module.

6. The methodology of the research and proposed work

Research methodology is a way to systematically solve the research problem. In research methodology, we not only talk of the research methods but also consider the logic behind the method we use in the context of our research study and explain why we are using a particular method or technique and why we are not using others. So that research results are capable of being evaluated either by researcher himself or others.

Qualitative, empirical, and exploratory approach has been used for the proposed research work. Several research papers and technical reports on fault management in distributed SDN controllers were studied during the literature review phase. In addition to these different network simulators were also explored. Mininet [6] simulator was chosen to implement the proposed model. It provides very good GUI, Openflow switch, controllers, flexibility for generating complex topologies, and analysis of the results.

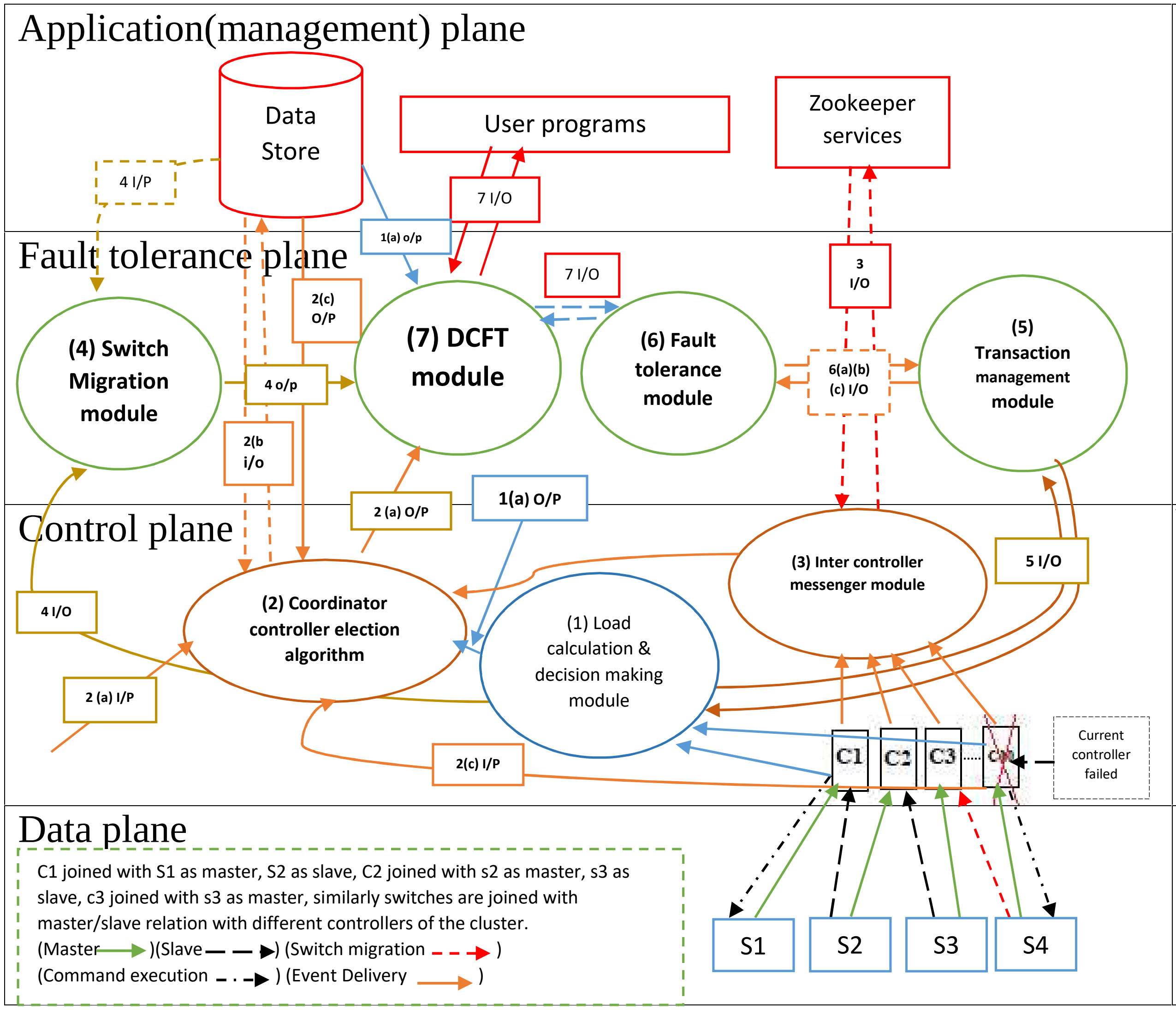
To solve the aforementioned research problem, our research work is divided into the following phases.

- Formulation of the research objectives and problem statement.
- Extensive literature survey.
- Designing the proposed system architecture of the DCFT model
- Designing of the algorithms
- simulation and results
- conclusion

7. Proposed DCFT (Distributed Controller Fault Tolerance) model:

We have named the our proposed research model as Distributed Controller Fault Tolerance (DCFT) model which contains data plane, control plane, and application (management) plane including one additional sub layer called fault tolerance plane in SDN stack, which is extended from application plane. Fault tolerance sub layer comprises different modules such as switch migration module DCFT module, fault tolerance module, and transaction management module.

7.1 Architecture of the DCFT model


Application(management) plane
Data Store
User programs
Zookeeper services
4 I/P
7 I/O
1(a) o/p
Fault tolerance plane
2(c) O/P
(7) DCFT module
7 I/O
(6) Fault tolerance module
3 I/O
6(a)(b) (c) I/O
(5) Transaction management module
(4) Switch Migration module
4 o/p
2(b i/o
2 (a) O/P
1(a) O/P
Control plane
(3) Inter controller messenger module
5 I/O
4 I/O
(2) Coordinator controller election algorithm
(1) Load calculation & decision making module
2 (a) I/P
2(c) I/P
C1
C2
C3
CN
Current controller failed
Data plane
S1
S2
S3
S4
(Master)(Slave) (Switch migration)
(Command execution) (Event Delivery)

C1 joined with S1 as master, S2 as slave, C2 joined with s2 as master, s3 as slave, c3 joined with s3 as master, similarly switches are joined with master/slave relation with different controllers of the cluster.

Fault tolerance sub layer will be inserted between application (management) plane and control plane.
DCFT module, main module of the proposed model, saved current state of the system, it will also save changes/updates by switch migration, fault tolerance modules. Publish updates/sync controllers with the help of inter controller messenger, It will receive input from the user program about state of the controller, output will be informing user program about fault management.
Switch migration module provide load balancing and avoid failure on overloading. Overloaded switch will be migrated to least loaded controller, selected from arraylist, maintained in store.

Control plane did routing task based on the policy decision of management plane, load of each controller calculated, coordinator controller will be decided, inter controller messanger will publish update/sync controllers.
C1, C2, C3,…, C10 are the controllers of the cluster. C10 is current coordinator, If C10 failed, orphan switches of C10 will be migrated to C3(least loaded), new coordinator is C9, controller having highest priority

Data plane comprises switches with flow table entries, packet forwarded as per the routing decision of the controller.

Figure 1 Architectural diagram of DCFT model-position of all the modules in SDN stack [8]

Application plane	It will store SDN applications, perform event/command ordering, command execution, distributed log records are shared via this plane, zookeeper coordinating service also installed at application plane.
Fault tolerance plane	3 i/p: subscribe updates of the state of each controller of the cluster. 3 o/p: publish update of the state of each controller of the cluster, sync state of the controller, use zookeeper for internal communication. 4 i/p: overloaded controller from load calculation module 4 o/p: select least loaded controller from arraylist maintained in distributed db. 4 o/p: select least loaded controller from arraylist maintained in store, failure of the any controller of the cluster due to any reason, orphan switches need to migrate to least loaded controller selected from array list maintained in distributed log. 7 i/p: It will received updates from SDN applications about state of the controllers. DCFT module save the current state of the system, it will also reflect the changes/updates done by switch migration/fault tolerance/transaction management module. Inter controller messenger module provides coordination services through zookeeper via DCFT module. While fault occurs, it will tolerate fault through fault tolerance and transaction management module. 7 o/p: It will inform SDN applications runs on application plane about occurrence and management of fault. 5 i/p: Events generated by switches on receiving packet or states of the port changes 5 o/p: call transaction management module, provide ACID(atomicity, consistency, isolation and durability properties with NIB, Optimistic concurrency control and distributed log. 6 (a) i/p: coordinator (master) controller failed before replicating received event in distributed log. 6 (a) o/p: call transaction management module, slave controller receive and buffer all events, no events are lost, first new master must finish processing any events logged by the old master, events marked as processed have their resulting command filtered. 6 (b) i/p: Coordinator(master) controller failed after replicating the event but before commit request 6 (b) o/p: Event was replicated in the distributed log, the master that crashed may or may not have issued the commit request message. therefore new master must carefully verify if the switch has processed everything it has received, before resending the command and commit request. 6(c) i/p: coordinator (master) controller failed after sending commit request 6(c) o/p: since old master send commit request before crashing, the new master will receive the confirmation that the switch processed the respective commands for that event and will not resend them (guaranteeing exactly once semantics for commands)
Control plane	2(a) i/p : Distributed SDN controller from the cluster 2(a) o/p : coordinator controller decided, informed to DCFT module. 2(b) i/p: module stores load information, perform load balancing and routing of packets 2(b) o/p: store switch controller mapping information 2(c) i/p: failure of the coordinator can be detected by its slave controller on timeout. 2(c) o/p: On failure of the coordinator, next coordinator will be elected from priority arraylist of coordinator maintained in distributed log. 1(a) i/p N- no of flow table entries, F-average message arrival rate, D-Propagation delay 1(a) o/p C_{load}= w1*N + w2*F+w3*D
Data plane	Forwarding packets as per the routing instructions received by controller.

Table 1 Events description of the position of modules in SDN stack

7.2 Deployment diagram of the DCFT model

- There are many approaches used to design a fault-tolerant SDN controller. The distributed-load approach target large-scale networks where a single controller has difficulties to manage the entire load and hence the operation has to be split into several domains with corresponding dedicated controllers. In this design, fault tolerance may be provided by making the appropriate load redistribution in the case of controller failures. In this way the controllers can act independently and in parallel, as long as they share the common network view, i.e. same network information base (NIB) is used for the shared data store. If consistency prioritized, performance degradation is unavoidable.
- While in master-slave SDN controller is a simpler concept in which only master controller is incharge of all decisions. In our model coordinator controller is taken as master controller, while the slave controllers are used to provide fault tolerance. In flat model, every distributed controller can acts as not only ordinary controller but also a coordinator controller. So it just needs one network transmission for gathering load and push commands to another controller, as a result, decision delay of the packet is reduced in this case[8]. In addition to this, all the controllers share the same consistent network view and locally serve request without actively contacting any remote node, thus minimizing flow setup up delay. This approach may be implemented in small or medium scale networks. The main challenge is provide consistency between network image (NIB) between master and slave controllers which is implemented through common data store. Our model provides fault tolerance through transaction management. Transaction management module provides consistency guarantee of the common data store with respect to packet loss, failure recovery, and overhead. Failure recovery will be achieved through path migration. Overhead will be increased with respect to number of rules update per phase.

- Two controllers are taken from different domains of the distributed SDN controller in figure 2 for demonstration purpose. Transaction management for fault tolerance is discussed in detail in section 7.8. In distributed SDN, events are generated due to the changes in port or packet sent from switch to controller. Figure 2 shows (1) The events are generated from switches and delivered to controllers (2) all the events are ordered and its updates are synchronized with other slave controllers using zookeeper and log records are stored in distributed data store (3) The controller run multiple application that received events and process it. (4) After processing controller send commands to the switches in reply to each event. This cycle repeat itself in multiple switches across the network as needed. Other modules of the model are described below.

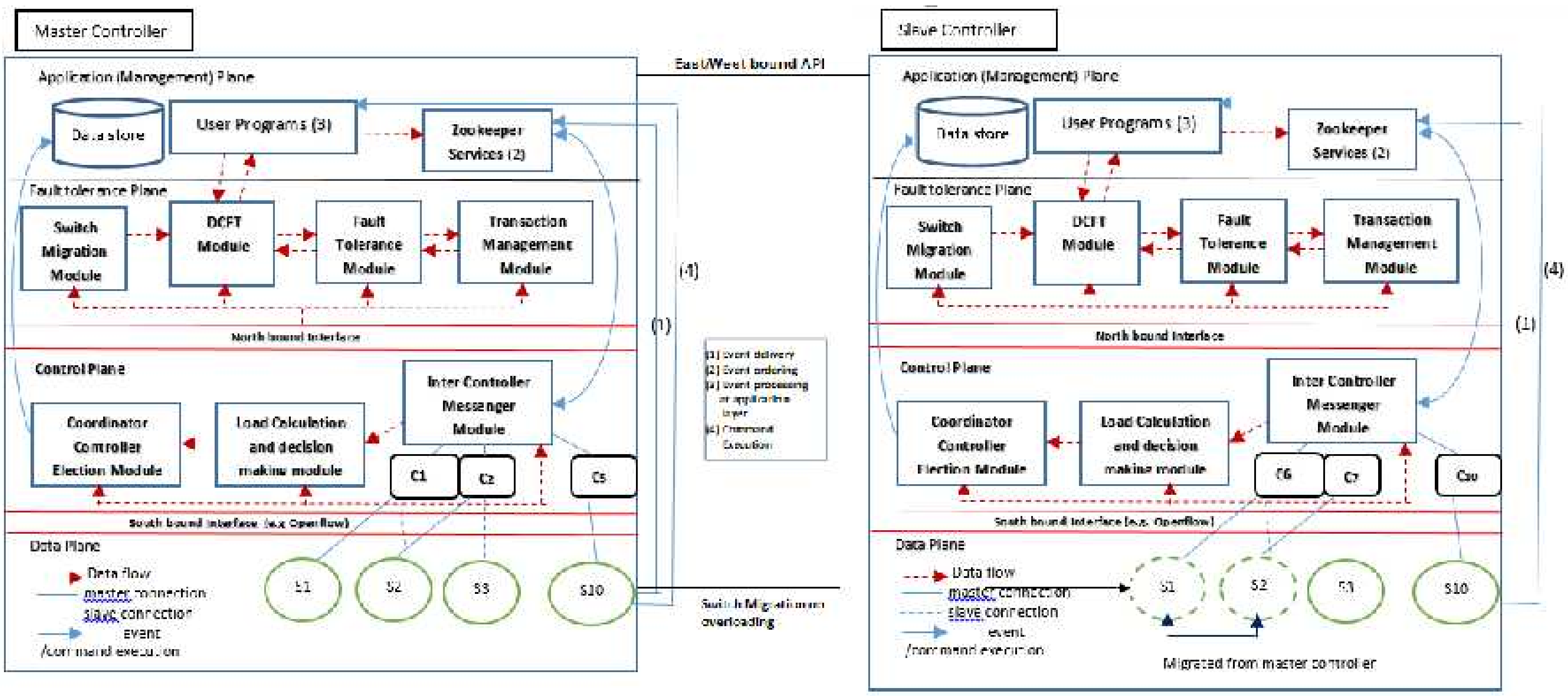


Figure 2 Deployment diagram of the DCFT model [13] [18]

) **Working of each modules**

7.3 Coordinator controller Election module:

This module [8] will be working on control plane, as shown in figure 1. Coordinator controller of the cluster 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. Our model follows Master/Slave connection between switch to controller. As discussed in literature survey, in Openflow 1.2 only single Master controller, many slave/equal role of the controllers are allowed in the cluster. The controller’s IP address recognizes each controller and their role, 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 including average message arrival rate and propagation delay.

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 failed controller and take 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 c1 as the default coordinator.

Algorithm 1: Coordinator controller election algorithm

```
1. pollTime=5 seconds // Election class is polled every "pollTime" seconds such that it checks if a new coordinator
    present in the network.
2. Once the coordinator specified and the follower's role decided, the current coordinator is used to managing network vide
    publishing and subscribing updates across all nodes.
3  pollTime=5 seconds, timeout=6 seconds
4. First try to get coordinator if (coordinator==none) then coordinator=controllerID1; timeoutFlag=true
5. else if (coordinator.equals (ControllerID)) then
6. roll-based function such as initiates publish/subscribe by the coordinator,
   //publish means ask all the nodes to call publish hook, subscribe means ask all the nodes to subscribe to updates
    from all other nodes as well as by calling this in a loop.
7. There are different possible states of the controller during the controller can be during the election process.
8. switch (current state) // current state of the controller
9. {
10. case CONNECT: Network block until the majority of nodes connected
11 case ELECT: check for the new node to connect to, and refresh socket connection, ensure the majority of nodes connected
otherwise goto CONNECT state.
12 start election if the majority of the node connected.
13 once coordinator has confirmed by-election it proceeds to coordinate or follow state.
14 case SPIN: This is resting state of coordinator after the election

     CheckForcoordinator: This function ensures that there is only one coordinator, set for the entire network, none or multiple
     coordinators causes it to set the current state to ELECT. It is based on predefined arraylist of controllerID. [2, 1, 4, 3] =>
     means that controller 2 will be considered first for election, if it fails we fall back to controller 1 as coordinator, and then
     controller 4 and so on.
15. case COORDINATE: This is resting state of coordinator after election keep sending heartbeat message and
          receive a  majority of acceptors otherwise goto ELECT state
16.}
17. check for only "one" coordinator in the network.
18. Ask each node if they are the coordinator, all the nodes should get an ACK from only one of them, if not reset
     the coordinator.
19. //Election performed.
20.   if (connected controllers >= 2) then
21.       if (elected coordinator present == true)
22.                 if (no of coordinator ==1)
23.                   commit; coordinator elected as controllerID=1
24.                 else
25.                   call checkForCoordinator function.
26.       else
27. Check for new node to be connect and controller having highest controllerID will become coordinator.
28. Nodes joined after election:  It follows the current coordinator
29. Nodes joined before election: It participates in current election process, coordinator will be elected from current
     active and configured controller.
30. Nodes joined during the election: It waits for election to be completed, does not participated in an election, start following
     elected coordinator after the election.
```

7.3.1 Simulation

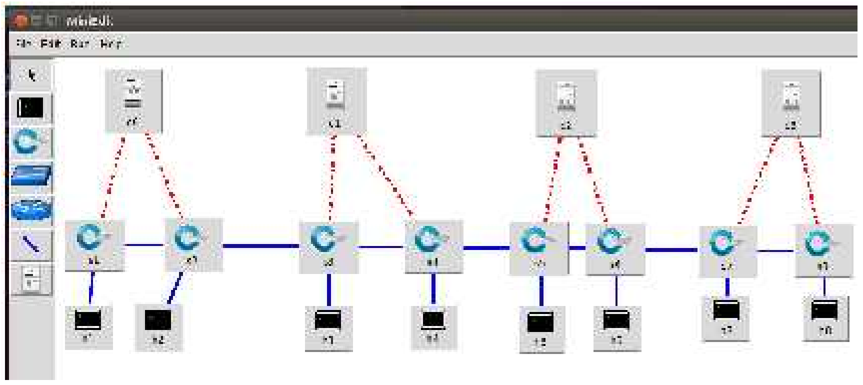


Figure 3. The topology used in the experiment

Traffic Sequence	Source	Destination
T1	H1	H4
T2	H3	H7
T3	H1	H8

Table 2 Traffic design used in the experiment

Without Coordinator controller							With Coordinator controller					
Traffic number	Packet delay (ms)	communication overhead (packet/s)		Throughput (packet/s)			Packet delay (s)	communication overhead (packet/s)		Throughput (packet/s)		
		Switch-controller	Controller-controller	C1	C2	C3		Switch-controller	Controller-controller	C1	C2	C3
T1	58.1	4034	3522	42332	42132	37934	28.66	4134	3623	54332	54123	49911
T2	68.9	3742	3012	39832	44445	35445	26.74	3942	3563	51834	56443	47452
T3	55.7	2343	2023	35445	42454	38832	24.20	2673	2642	51434	54463	50854

Table 3 Comparison of with coordinator/without coordinator controller in the cluster

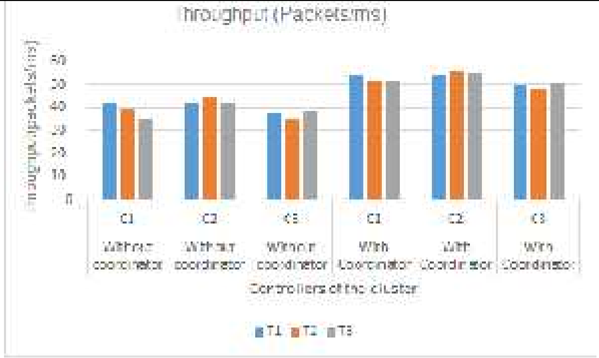


Figure 4 Throughput of different traffic sequences

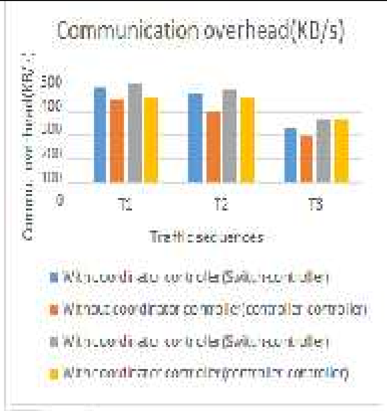


Figure 5 Communication overhead of different traffic sequences

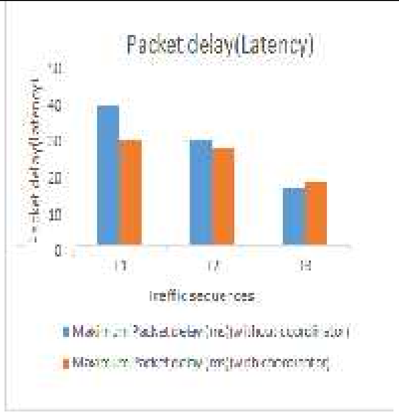


Figure 6 Packet delay(Latency) of different traffic sequences

This module proposed a coordinator controller election algorithm in the cluster of distributed SDN controllers. Our primary research goal is to propose the DCFT (Distributed Controller Fault Tolerance) model to provide fault tolerance through load balancing in the distributed SDN controller. The coordinator controller election is one module of the model. To provide a fault tolerance mechanism in the distributed SDN controller cluster, one additional fault tolerance sub-layer will be added in the SDN stack by extending an application plane of SDN. Four floodlight controllers are taken for the simulation. The result of the simulation will be tested for three different traffic sequences as shown in table 2. Comparison of with coordinator controller and without coordinator controller is demonstrated by table 3. With the coordinator, Considering an average of all three traffic sequence, the throughput of the cluster will be increased by 22.63%, packet delay(latency) will be decreased by 56.13% and communication overhead will be increased between switch-controller is 7.09 KB/s and between controller-controller is 23.47 KB/s. So by introducing coordinator controller in the distributed SDN controller, consistency, reliability, scalability of the system are improved at the cost of communication overhead.

7.4 Internal controller messenger module:

There are two ways to acquire network information from other SDN controllers. Polling and, Publish, and Subscribe [7]. This module opt publish/subscribe method as shown in figure 1.

(a) Polling: Each SDN controller periodically requests for new network information from other controllers in the cluster. For instance an SDN request for new switch information at every 10 minutes. It will execute the request periodically even when there is no update happen in the other controller. Thus it may receive the same network information as the last one. Therefore this method is not efficient [8].

(b) Publish and subscribe: Each SDN controller can publish/subscribe the network information from other controllers in the cluster. For instance controller c1 needs network information from neighboring controller c2, the controller c1 can subscribe the switch information from c2. In this case, c1 acts as a subscriber, and c2 acts as a publisher. Later c2 will notify c1 when there is a change regarding the switch information in the domain. In this case controller, c2 will notify c1 only when there is a change, therefore, this method is more efficient for our model [7].

Internal controller messenger 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. Distributed coordination service such as zookeeper [9] glues cluster of the controllers to share the information about a link, topology, etc. it's used for updating the status of the controllers.

7.5 Load Calculation and decision-making module

All the controllers including coordinator controller calculate its own load and send load information to the coordinator controller as shown in figure 1 in control plane. 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 an important 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) [10].

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

Where w1, w2, and w3 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.

(a) 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.

(b) Decision making 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 set the initial load balancing rate to 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 [11].

(c) 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 be migrated.

Load_of_Switch_to_Migrate ≤ CT – Load_of_Target

CT= Controller capacity (packets/sec)

Authors [12] 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.

7.6 Switch Migration Module

On failure of the controller, orphan switches need to be migrated to other controller. Our proposed switch migration algorithm [5] 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 selected 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 bear 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.

Switch migration occurs in three situations of failover. (1) Coordinator controller failure (2) ordinary controller failure (3) Load imbalance. Pseudocode for three conditions is as follows.

Algorithm: 2 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 the least loaded controller from the distributed database
        if (capacity of least loaded controller > threshold) then
        call switch migration module and migrate switches
        else
                migrate few switches up to 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
```

7.6.1 Simulation results

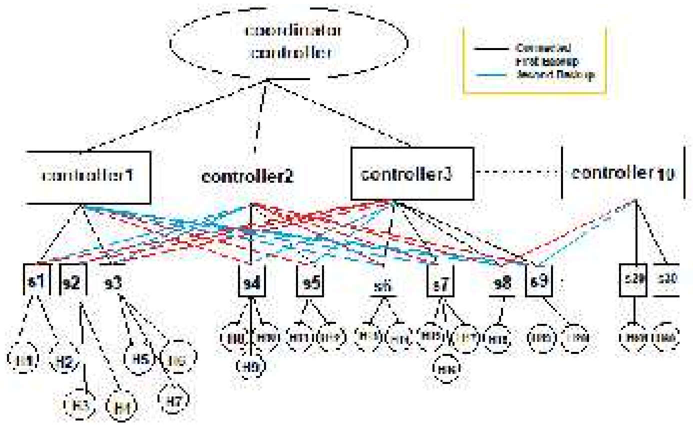


Figure 7. The logical perspective of the topology used in a simulation

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

Table 4 Traffic designs used in experiment

We use hping3 to generate TCP flows to simulate the dispersal of network traffic the average flow requests The average packet arrival rate 38000 flow/s. Flow counts of different topologies are shown in figure 8(a),8(b), and 8(c). 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. we set no of switches managed by one controller is from 2 to 10. All the simulations run for 2 hours readings are noted at every 20 minutes. Consider the topology shown in figure 7. DCFT model takes interruption among switches and their associated controllers to curtail the response time.

Before switch Migration									After switch Migration (At 15 second C2 failed)						
	Traffic number	Maximum Packet delay (ms)	Communication overhead (packet/s)		Throughput(packet/s)-Flow request processed by each controller				Maximum Packet delay (ms)	Communication overhead (packet/s)		Throughput(packet/s)- Flow request processed by each controller			
			Switch - controller	Controller-controller	C1	C2	C3	…C10		Switch - controller	Controller-controller	C1	C2	C3	…C10
Custom topology	**T1**	**40.1**	**4233**	3722	32432	33824	31106	32942	**24.27**	**5231**	**4324**	**43190**	**41245**	**42495**	**42346**
	T2	**30.9**	**3944**	3213	34123	34834	31106	31113	**27.32**	4952	**3812**	43505	42156	43345	44428
	T3	**16.8**	**2542**	2121	36453	35654	33134	32122	**12.21**	2672	**2567**	42062	43984	40674	43542
Abilene topology	**T1**	**32.5**	**5121**	3282	33421	36127	32215	33826	**20.41**	6173	**3728**	44662	441584	43473	41158
	T2	**26.8**	**4532**	3921	35143	31462	33143	31155	**14.82**	**5621**	**4374**	**45824**	**42794**	**42785**	**42862**
	T3	**14.3**	**2361**	3125	37627	35122	34181	34177	**10.24**	3378	**3486**	43175	43526	42374	43094
Internet 2 OS3E	**T1**	**42.5**	**6202**	2266	34324	36102	35217	32185	**30.46**	7254	**2844**	44648	42345	43116	43187
	T2	**35.8**	**3561**	2804	35142	34123	36184	31177	**23.64**	4523	**3415**	45421	43561	44134	44051
	T3	**18.6**	**2681**	2192	3353	37130	35150	31151	**13.65**	**3744**	**2841**	**42646**	**44615**	**42105**	**43815**

Table 5 Maximum packet delay(ms), Communication overhead and throughput in DCFT model (before/after switch migration)

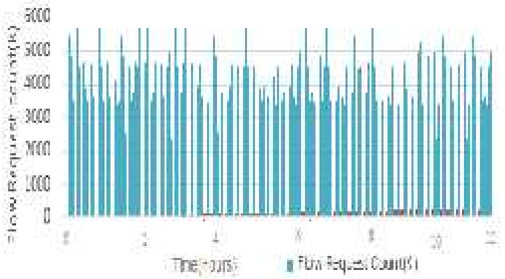


Fig 8(a) Network traffic in custom topology

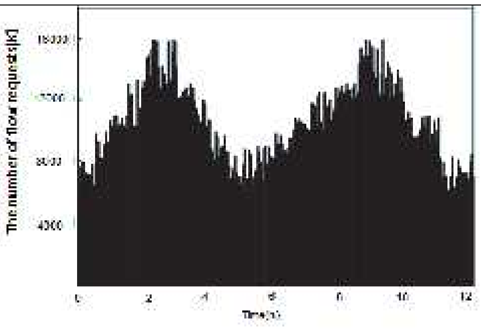


Fig 8(b) Network traffic on Abilene topology[37]

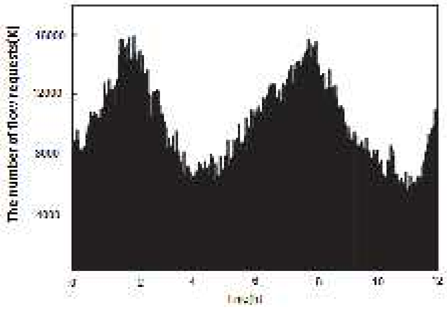


Fig 8(c) Network traffic on Internet 2 OS3E topology [38]

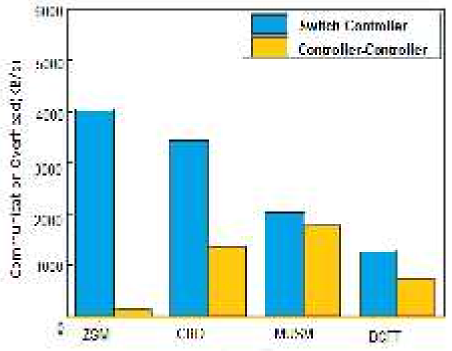


Fig 9 (a) Communication overhead in custom topology

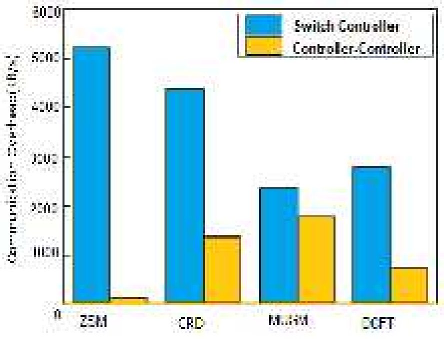


Fig 9(b) Communication overhead in Abilene

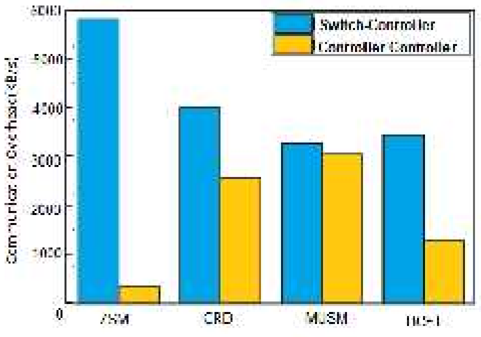


Fig 9(c) Communication overhead in OS3E

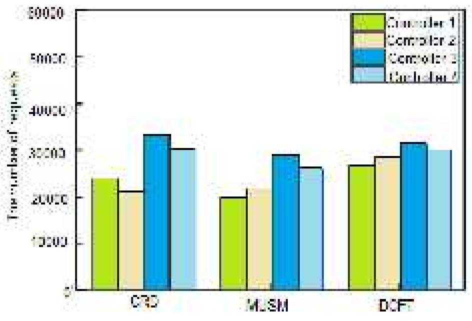


Fig 10(a)Throughput- Request processed by each controller – custom topology

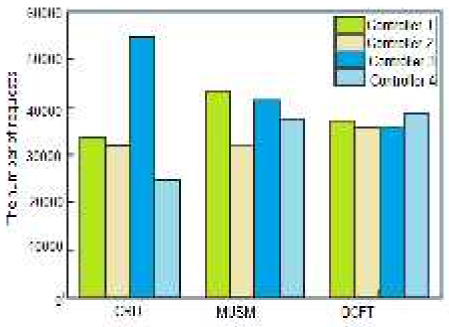


Fig 10(b) Throughput- Request processed by each controller- Abilene

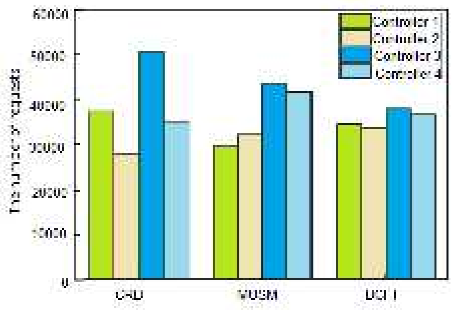


Fig 10(c) Throughput-Request processed by each controller- OS3E

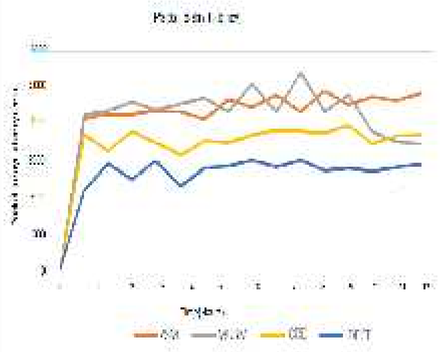


Fig 11(a) Packet delay(latency) in custom topology

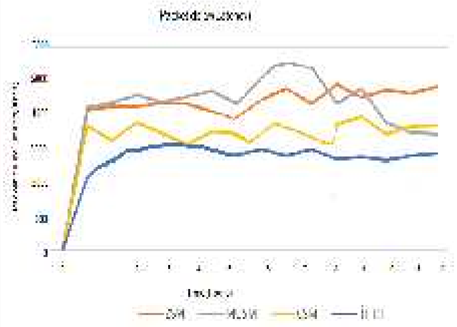


Fig 11(b) Packet delay(latency) in Abilene topology

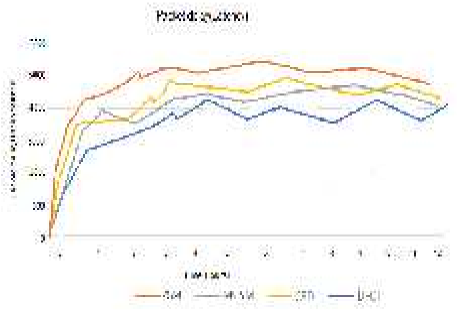


Fig 11(c) Packet delay(latency) in OS3E topology

7.6.2 Conclusion

In this section, we did a study of fault tolerance in the distributed controller with software defined networking using switch migration. Switch migration algorithm reveals cases of coordinator controller failure, ordinary controller failure, and load imbalance. Ten controllers are taken in the custom topology. On failure of coordinator controller, how orphan switches are migrated to the least loaded controller with the help of openflow commands were explained in detail. Simulation analysis performed with series of experiments performed using traffic patterns (Table 4), on custom topology, two well-known Abilene[14] Internet 2 OS3E[15] topologies ten controllers along with coordinator controller Communication overhead, controller load balance rate(throughput-flow request count processed by each controller), packet delay were used as evaluation indexes. Table 5 shown performance evaluation of only four controller out of ten to reduce the complexity. It was found in figure 11 that DCFT model reduces packet delay by 30.84 %. From figure 9, Average communication overhead was increased between switch-controller by 19.43 % and controller-controller is by 15.61%. From figure 10 throughput is increased 25.37 % in custom topology, 19.04% in Abilene and 19.03% in Internet 2 OS3E topology. So average throughput will be increased by 20.65%. It is concluded that by reducing packet delay and increasing throughput, DCFT model contributes better in fault tolerance in the distributed control plane despite of sensibly increase in communication overhead introduced by coordinator controller.

7.7 Flow of the Fault Tolerance module

In an SDN setting, switches generate events (e.g., when they receive packets or when the status of a port changes) that are forwarded to controllers. The controllers run multiple applications that process the received events and may send commands to one or more switches in reply to each event. This cycle repeats itself in multiple switches across the network as needed.

In order to maintain a correct system in the presence of faults, states of switch and controller must handle consistently. To ensure this, the entire cycle presented in figure 12 is processed as a transaction: either all or none of the components of this transaction are executed. This means that (i)the events are processed exactly once at the controllers,(ii) all controllers process events in the same (total) order to reach the same state, and(iii)the commands are processed exactly once in the switches.

Because the standard operation in openflow switches is to simply process commands as they are received, the controllers must coordinate to guarantee the required exactly-once semantics.

Authors in [16] do not need this coordination because the (modified) switches can simply buffer the commands received and discard repeated commands (i.e., those with the same identifier) sent by the new controller.

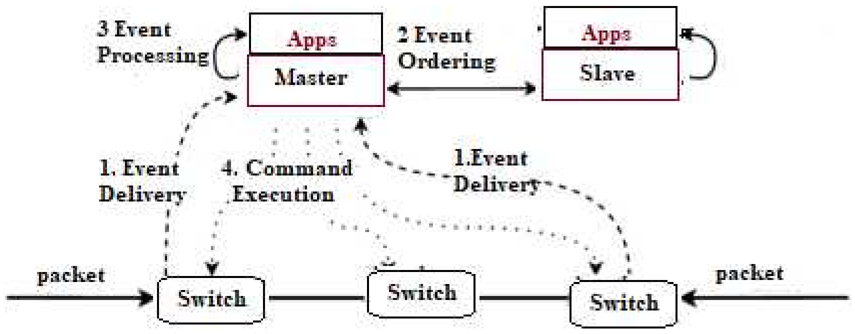


Figure 12: Control loop of (1) event delivery (2) event ordering (3) event processing and (4) command execution. Events are delivered to the master controller, which decides a total order on the received events. The events are processed by the application in the same order in all controllers. Application issue commands to be executed in the switches.

By default, in openflow, a master controller receives all asynchronous messages (e.g., OFPT_PACKET_IN), whereas the slaves' controllers only receive a subset (e.g., port modifications). With this configuration, only the master controller would receive the events generated by switches. There are two options to solve this problem.

One is for slaves to change this behavior by sending an OFPT_SET_ASYNC message to each switch that modifies the asynchronous configuration. As a result, switches send all required events to the slaves. Alternatively, all controllers can set their role to EQUAL. The openflow protocol specifies that switches should send all events to every controller with this role.

Then, controllers need to coordinate between themselves who the master is (i.e., the one that processes and sends commands). We have opted for the second solution and use the coordination service for coordinator election amongst controllers.

7.7.1 Fault cases of the DCFT module

One of our research goals is to build a fault-tolerant distributed SDN control plane to assure the correctness of logically centralized controllers. We had taken three fault cases and discuss the process of fault tolerance for these cases.

When the master controller fails, the slave controllers will detect the failure(by timeout) and run a coordinator election algorithm to elect a new master must send a role request message to each switch, to register as the new master. There are three main cases where the master controller can fail.

1. Before replicating the received event in the distributed log.
2. After replicating the event but before sending the commit request.
3. After sending the commit request message.

In this section, we summarize how the mechanisms employed by our model fulfill each of these necessary requirements.

(a) Exactly once event ordering: Events cannot be lost (processed at least once) due to controller faults nor can they be processed repeatedly (they must be processed at most once).
Contrary to the model of [16], DCFT model does not need switches to buffer events neither that controllers acknowledge each received event to achieve at least once event processing semantics. Instead, DCFT model relies on switches sending the generated events to all (f+1) controllers (considering that the system tolerates up to f crash faults) so that at least one will be known about the event. Upon receiving these events, the master replicates them in the shared log while the slaves add the events to a buffer.

As such, in case the master fails before replicating the events, the newly elected master can append the buffered events to the log. If the master fails after replicating the events, the slaves will filter the events in the buffer to avoid duplicate events in the log.

This ensures at-most-once event processing since the new master only processes each event in the log once. Together, sending events to all controllers and filtering buffered events ensures exactly-once event processing.

(b) Total event ordering: To guarantee that all controller replicas reach the same internal state, they must process any sequence of events in the same order. For this, model of [16] and DCFT model rely on a shared log across the controller replicas (implemented using the external coordination service such as zookeeper [9]) which allows the master to dictate the order of events to be followed by all replicas. Even if the master fails, the newly elected master always preserves the order of events in the log and can only append new events to it.

(c) Exactly once command execution: For any given event received from one switch, the resulting series of commands sent by the controller are processed by the affected switches exactly once. Here, the model of [16] relies on switches acknowledging and buffering the commands received from controllers (to filter duplicates).

As this requires changes to the openflow protocol and to switches, DCFT relies on openflow bundles to guarantee transactional processing of commands. Additionally, the commit reply message, which is triggered after the bundle finishes, is sent to all controllers and thus acts as an acknowledgment that is independent of controller faults.

If the master fails, the new master needs to know if it should resend the commands for the logged events or not. A packet_out message at the end of the bundle acts as a commit reply message to the slave controllers. This way, upon becoming the new master, the controller replica has the required information to know if the switch processed the commands inside the bundle or not, without relying on the crashed master.

Furthermore, the new master sends a barrier request message to the switch. Receiving the corresponding barrier reply message guarantees that neither the switch nor the link is slow (because a message was received and TCP maintains FIFO order) and thus there is no possibility of the packet out being delayed. Therefore, the use of bundles that include a packet_out at the end, in addition to the barrier message ensures that commands will be processed by the switches exactly-once.

7.8 Transaction management module

Three fault cases are discussed with correctness for the consistency in fault tolerance module. Message bundling technique realizes atomicity, consistency. Many controllers are involved in distributed controller with their multiple events. Transaction management module (TMM) is introduced in the fault management plane, which resides between application plane and control plane. It is used for network update services in SDN applications and transactionally update the network. TMM’s interface called by the SDN applications to achieve durability, integrity, atomicity, and consistencies for inter-update isolation. TMM will compare network match field (flow headers) in the rules and solve conflict resolution. TMM takes SDN update from the many applications, devices the ACID execution of concurrent updates, and at the end binds the update to the network.

7.8.1 Realization of ACID properties of database in TMM

- Consistency: The module consist of API which takes care of the changes occurring due to any abnormal situation or unresolved situation.
- Atomicity: The module consists of an audit log which is used to monitor the updates in the network. The primary component called an atom can be traced and corrected.
- Isolation: Many transactions occur at the same time in the network which is managed in an unbiased manner and all the resources satisfied based on priority.
- Durability: Constant updates in the network can create integrity problems. Once updation takes place, it should not allow the state to change until proper authorization is available.

TMM module is inserted between SDN application plane and SDN control plane. It works as enhancement layer. Northbound API of the network applications, devises and executes the update to the network. The issues arising which need to be monitored and contained are listed below:

Issue 1: Classifying network changes and events occurring in network communication channel (data plane) is a major challenge.

While change of state continued in the network, simultaneously network state in the data plane also changes, it includes records of packet processing. It may lead to error. In networks, conflict resolution will be resolved through data plane events. It flows from data plane to control plane.

***Solution**:* Divide network states into determined and unstable states. Determined states are read/write by control planes with flow rules. While unstable states are read by control plane with flow statistics. As network state changes completes, it would monitor the changes and commit the updates, if failed reject updates.

Issue 2: Simultaneous changes can pose a great risk.

TMM should have interface and implementation to support the updates into stages. Two stage updates are used by TMM. Communication delay is introduced due to two stage updates in SDN switches. So order of the update may be affected.

Solution: TMM present new operator fence ( ) for SDN applications. It helps to divide two stages updates into bunches. Introducing the semantics of the stages into network updates. Read state of the switch in the same bunch would be read by the TMM. fence ( ) function would work in two stage and multi stage update using log module for SDN application

<u>*7.8.2 TMM architecture*</u>

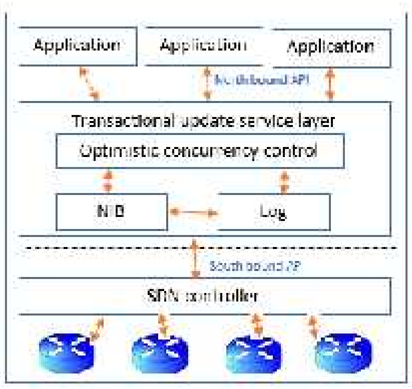


Figure 13 TMM architecture [17]

TMM is formed of a combination of three main tasks:

1. Control of Concurrency has Northbound API's.
2. NIB consisting of southbound API's.
3. Audit of Transactions.

<u>**Algorithm 3: 2 stage update**</u>

```
t= transaction ( )
first goal, other goal=acquire goal (other path)
for each goal in other goal
        t.write (goal)
end
t.fence ( )
t.write (first goal)
if t.commit ( ) failed then
        do somewhat
end
```

The algorithm demonstrate that the TMM functions are used to arrange network update. The example is 2 stage update. t.write ( ) and t.fence ( ) are used for the update transaction. Eventually, t.commit authorize the update and update it to NIB.

7.8.3 Evaluation of the Implementation

TMM will be implemented using floodlight SDN controller, with Intel I3 processor, 4 GB RAM. Mininet [6] will be used to emulate of network topologies in the implementation. Packet loss and failure recovery and overhead management done by the TMM will be demonstrated through simulation.

- ***Packet loss:*** TMM used fence for transactions, packet loss are monitored with and without fence. Online path migration is used in algorithm 3. Packet loss is monitored in both with fence and without fence. Experiment repeated several times it is observed that packet loss without fence is 984.6 +/- 523.3 and with fence, it is 9.2+/- 1.6. So with fence, number of packet drop reduces significantly.

- ***Failure recovery:*** On failure of SDN application or SDN controller, failure reported and failover performed to a new instance. Log module used to replay or reject update and check atomicity. READ, WRITE, VALIDATION actions are taken. Consider the topology used in figure 18. Updates having READ and INACTIVE status are rejected. WRITE stage updates are executed again and marked as INACTIVE. VALIDATION stage update are taken care by concurrency control. Topology shown in figure 14 will be used for the demonstration of failure recovery.

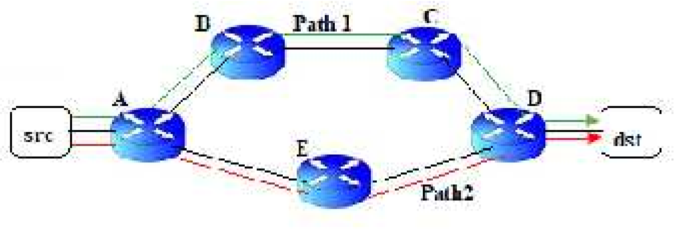


Figure 14 topology used in experiment

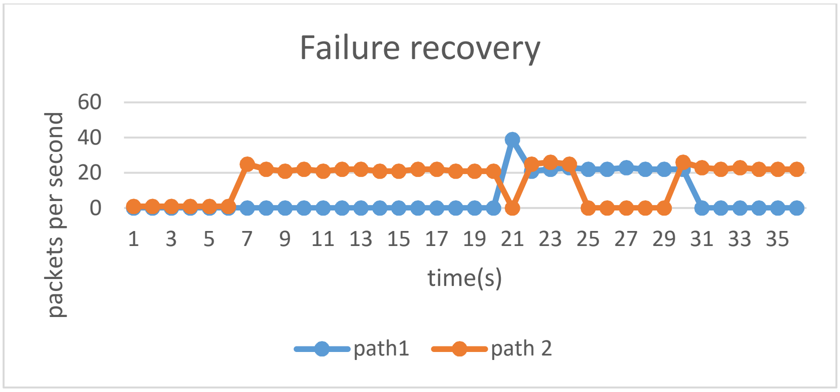


Figure 15 Failure recovery

Throughput of path 1 and path 2 will be monitored. There are few routing rules in the switches. At certain time controller reboots with few logs like <path 1, WRITE>, <path 2, START> shows path 1 is in WRITE stage and path 2 in READ stage. After restart path 1 setup completes and path relocation is discarded. Figure 15 shows path 1 and path 2 for packet sequence at different times. In path 1, at t=6s controller will receive 25 packets, and continued in receiving packets till its first reboot on t=19s. On restart again it will started from the last state from the log. Similarly, sequence monitored from path 2.

- ***Overhead measurement:*** It is measured by log module of the TMM. Experiment starts with no of rules are installed in few stages to a switch. Total no of rules against no of rules per phase and their installation time will be compared in with and without log module as shown in figure 20. I/O time and fence wait time are influencing total installation time. In one rule update, noticeable changes are not monitored in with and without log. If all the rules are updated in one phase, difference is noticeable in figure 24(c). Log module origins 18 % performance degradation where per phase 10 rules are updated.

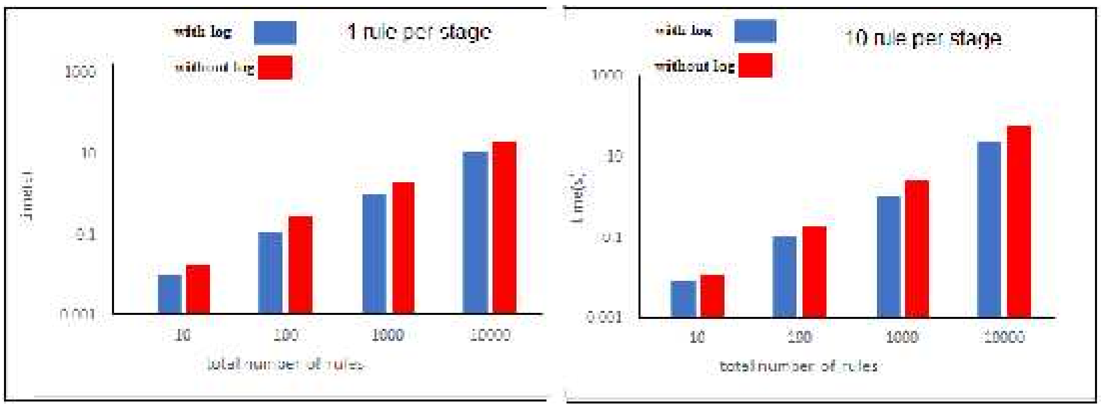


Figure 16 Overhead measurement (a) 1 rule per stage (b) 10 rule per stage

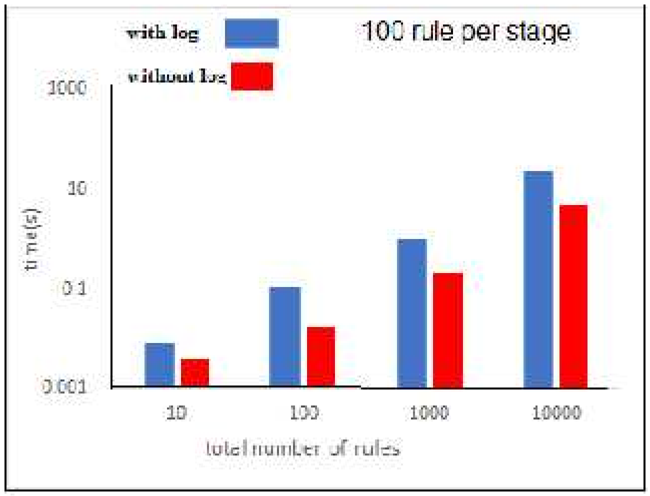


Figure 16 (c) Overhead measurement-100 rule per phase

7.8.4 Conclusion

This section discussed at large the applications and challenges involved in load balancing and related applications in SDN. TMM proves its credibility in providing storage, retrieval, and transparency in its operations. The parameters on which the comparison made include recovery rate, overhead measurement, integrity, and transparency. Detailed assessment shows TMM providing integrity, better failure recovery, and has limited overhead. Future work involves testing TMM in additional predefined and changed settings, for problems arising due to workload, network topology, and situations (e.g., failure, recovery), with the functional measurements.

7.9 DCFT module

One of the SDN principles is the logical centralization of the control, allowing a reduction of management complexity and network heterogeneity. Two possible approaches to achieve this are physically distributed control and physically centralized control. The former approach provides more flexibility, enabling clustering techniques and increasing resilience, however it requires coordination capabilities in the control plane. The latter option is less complex but faces scalability and resiliency issues since it represents a single point of failure. The usual approach is to maintain backup replicas that may take control of the network in case of failure. In both approaches, a controller failure causes loss of state. In order to guarantee service availability, control plane state must have some level of redundancy. Both controller failure and application failure may lead to state loss, thus they must be handled separately if they do not share the same storage.

DCFT module is the main module of the proposed model, saved current state of the system, it will also save changes/updates by switch migration and fault tolerance modules. Publish updates/sync controllers with the help of inter controller messenger, It will receive input from the user program about state of the controller, output will be informing user program about fault management.

8. Achievement with respect to objectives

We successfully proposed and implemented the DCFT model which

- Provides architecture of fault-tolerant distributed control plane.
- Provides more reliability, consistency, and scalability through coordinator controller election. Provide failover mechanism for a control plane using the coordinator election algorithm.
- Provides transactionally update service in SDN applications and transactionally update the network. Achieved consistency guarantee with respect to packet loss, failure recovery.
- Provides strongly consistent fault tolerant control plane for distributed SDN controller and behave the same as fault free distributed SDN for the user through fault tolerance and transaction management module.
- Provides switch migration algorithm for better utilization of controller resources and propose a novel load balancing model which helps to achieve better throughput, fault tolerance, reduction in load balancing rate at the cost of communication overhead.

) Evaluate the performance of the controller vide packet delay, throughput and packet loss, rate of load balancing, and communication overhead.

9. Conclusion of the Research work

This research work far reaching view on fault management in distributed SDN controller. Distributed controllers with its properties are studied. Thesis presents design choice of distributed SDN controllers and its analysis based on switch to controller connection, network information distribution strategy, controller coordination strategy, and In-band vs. Out-of-band connection strategy.

Thesis surveyed through classification of SDN controllers vide their adaptability, documentation, design choice (hierarchical/flat). Main concentration given on the faults generated because of overloading on the controllers. Controllers are failed due to overloading and their orphan switches need to migrate to the underloaded controller. Prior work done on switch migration technique studied and finally one additional fault tolerance plane will be derived from application plane and inserted between application plane and control plane in the SDN stack.

Model represents SDN stack, coordinator controller election module would be elected coordinator controller of the cluster, and coordinator will decide next coordinator on the failure of the current coordinator. For the sample, three fault cases are discussed in fault management module. Transaction module provides an interface called by the SDN application to achieve ACID properties. It will take SDN update from many applications, devises the ACID execution of the concurrent updates, and at the end binds the updates to the network. We evaluate performance of the model (1) with and without coordinator controller (2) before and after switch migration and (3) in transaction management module. All the cases are tested for custom topology and other two well-known topologies Abilene [14] and Internet 2 OS3E [15]. On averaging of all cases, we concluded that model reduces packet delay by 30.84%, rate of load balancing, and eventually achieved higher throughput by 20.65 % at the cost of communication overhead (between switch-controller-11.47 % and controller-controller 11.10%)and generating robust distributed SDN controller.

10. Future work

Future issues related with distributed SDN controllers are (1) Standardization of protocol in the east-west control plane. (3) Standardization of northbound interface development (4) Dynamic load balancing mechanism with integration of AI techniques with SDN. (5) Integrating SDN/NFV with a distributed SDN controller.

11. Published papers

1. Lakhani G., Kothari A., (2019). Distributed Controller Fault Tolerance Model (DCFT) Using Load Balancing in Software Defined Networking, International Journal of Computer Engineering and Technology, ISSN: 0976-6367, Volume 10 Issue 2, 2019, pp. 215-233.

2. Lakhani G., Kothari A., (2019). Comparison of performance of Distributed Controller Fault Tolerance(DCFT) module using load balancing in software defined networking in well-known topologies, International Journal on emerging technologies E-ISSN 2249-3255, P- ISSN 0975-8664, Volume 10, Issue 2, Page No pp.136-146. **(Scopus Indexed)**

3. Lakhani G., Kothari A., (2019). "Fault management through load balancing in Distributed controller with SDN – A review", IJRAR - International Journal of Research and Analytical Reviews (IJRAR), E-ISSN 2348-1269, P- ISSN 2349-5138, Volume.6, Issue 2, Page No pp.849-868.

4. Lakhani G., Kothari A., (2020). Fault management in Distributed SDN controller through transaction management. International Journal of Advanced Science and Technology, ISSN: 2005-4238, 29(7), 1030 - 1036. **(Scopus/Elsevier indexed)**

5. Lakhani G., Kothari A., (2020). Coordinator controller election algorithm to provide failsafe through load balancing in Distributed SDN control plane, Proceedings of the 1st Springer CCIS series conference, ISSN: 1865-0929, COMS2, March 2020. **(Scopus indexed)**

6. Lakhani G., Kothari A., (2020). Fault management through load balancing in Distributed SDN controller-A review. Wireless Personal communication, ISSN: 0929-6212, E-ISSN: 1572-834X, **(Scopus indexed)**

7. Lakhani G., Kothari A., (2017). Topology discovery in distributed SDN controller, Journal of Maharaja Sayajirao University of Baroda (Science and Technology) –ISSN: 0025-0422, pp. 9-15.

References:

[1] 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.

[2] Koponen, Teemu, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan et al. "Onix: A distributed control platform for large-scale production networks." In OSDI, vol. 10, pp. 1-6. 2010.

[3] 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.

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

[5] Lakhani, G., Kothari, A., "Distributed Controller Fault tolerance model using load balancing in software defined networking", International Journal of Computer Engineering & Technology (IJCET) Volume 10, Issue 2, pp. 215-233 2019.

[6] 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.

[7] Oktian, Yustus Eko, et al. "Distributed SDN controller system: A survey on design choice." computer networks 121 (2017): 100-11.

[8] Lakhani, G., Kothari, A., "Coordinator controller election algorithm to provide failsafe through load balancing in Distributed SDN control plane", Proceedings of the 1st Springer CCIS series conference, COMS2, March 2020. **(Scopus Indexed)**

[9] 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

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

[11] 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.

[12] 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.

[13] Lakhani, G., Kothari, A., "Comparison of performance of Distributed Controller Fault Tolerance model using load balancing in software defined networking in well-known topologies", International Journal of Emerging Technologies, Volume 10 Issue 2, pp. 136-149, 2019. **(Scopus indexed)**

[14] Abilene Topology [online] available: "http://www.topology-zoo.org/files/Abilene.graphml" accessed online on 19/10/2018)

[15] Internet 2 OS3E topology [online] available at
http://www.internet2.edu/media/medialibrary/files/.graphml" accessed online on 19/10/2018)

[16] 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.

[17] Lakhani, G., Kothari, A., (2020). Fault management in Distributed SDN controller through transaction management. International Journal of Advanced Science and Technology, 29(7), 1030 - 1036.

[18] Lakhani, G., Kothari, A. Fault Administration by Load Balancing in Distributed SDN Controller: A Review. Wireless Personal Communication (2020). ISSN: 0929-6212, E-ISSN: 1572-834X.
https://doi.org/10.1007/s11277-020-07545-2***(Scopus indexed).**

[illegible] **(Dr. Amit Kothari, Supervisor)**

[illegible] **(Dr. Bhushan Trivedi, DPC Member)**

[illegible] **(Dr. Satyen Parikh, DPC Member)**