Coordinator Controller Election Algorithm to Provide Failsafe Through Load Balancing in Distributed SDN Control Plane

Gaurang Lakhani\textsuperscript{1} and Amit Kothari\textsuperscript{2}

\textsuperscript{1} CE/IT, Gujarat Technological University, Ahmedabad, India
\texttt{gvlakhani1@gmail.com}
\textsuperscript{2} Accenture, Bengaluru, India
\texttt{amitdkothari@gmail.com}

Abstract. SDN provides flexibility, centralized control, and cost-cutting. SDN decouples the main two functions of a traditional network viz. packet switching and routing. The switch is responsible to forward packets only without any concern about routing decisions or any security checks in SDN. The control plane is responsible for routing decisions. Distributed SDN solves issues of availability and scalability, which are the weakness of a centralized or single control plane. Fault Tolerance and load balancing are the key features of any network. A brief description of our primary research goal of the proposed DCFT (Distributed Controller Fault Tolerance) model is given here. A major focus is given on a coordinator controller election algorithm to provide failsafe through load balancing in the SDN control plane. Implementation of the said algorithm verified with floodlight controller with and without a coordinator, Implementation results reveals that throughput and communication overhead will be increased with coordinator controller, packet delay (latency) will be decreased with coordinator controller.

Keywords: Software Defined Networking · Fault tolerance · Switch migration · Coordinator election · Load balancing · Data plane · Control plane

1 Introduction

Software defined networking (SDN) is innovative method in network management and enable innovation in networking. Current traditional network are multifaceted and difficult to manage especially in the light of changing routing and other quality of service demands by the administrators. SDN controllers are considered as “brain” of the control plane. Controllers connects with network devices through south bound interfaces (SBI) such as openflow protocols. 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 [1].
SDN have flow tables. Flow means combinations of match criteria and its associated actions in the set of packet field. All the packets of the same flow will have to follow the same service polices at the forwarding devices. The flow abstraction permits uniting the performance of the different types of network devices such as routers, switches, firewalls, and middleboxes. Flow programming empowers extraordinary flexibility, limited only to the capabilities of the implemented flow tables. The separation of the control plane and data plane can be appreciated by means of 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 plays out specific activity on traffic. Contingent upon the rules installed by controller application, an openflow switch trained by controller to behave like a router, switch, firewall or perform any other roles [1].

Lack of flexibility, centralized control, and cost are such limitations of traditional networks. Software Defined Networking is a novel way to deal with these issues of traditional networking. SDN decouples the main two functions of a traditional network viz. packet switching and routing. In SDN switch is responsible to forward packets only without any concern about routing decisions or any security checks. The control plane is responsible for routing decisions. Control plane implemented in a distributed fashion where multiple controllers act with synchronization. Openflow [2] APIs are used by SDN to set up forwarding rules and collect statistics in the data plane, which allows controller software and data plane hardware to progress independently. With SDN physical connectivity between two endpoints does not guarantee they can interconnect with each other – the underlying (logical) communication graph be contingent on the network policies reflected by the flow entries installed by the controller. For scalability and reliability, the logically centralized control plane (“network OS”) is often appreciated via multiple SDN controllers (see Fig. 1) forming a distributed system. OpenDayLight [4] and Open Networking Operating System (ONOS) [5] and are two such Network OS examples supporting SDN controllers for high availability.

The main focus of this paper is to propose coordinator controller election algorithm to provide load balancing among distributed SDN controller, provide consistency reliability, scalability, transparency, and immediate deployability in distributed SDN controller using state replication at the cost of little degradation in performance.

![Fig. 1. SDN control plane setup](image-url)
The remaining paper is arranged as follows: in Sect. 2 previous paper is discussed, the Objective of the proposed work is given in Sect. 3. Section 4 shows the proposed coordinator election algorithm with a brief description of the DCFT model Evaluation and results in Sect. 5, the conclusion is included in Sect. 6.

2 Related Work

2.1 Architecture of Distributed SDN Control Plane

Our prime research goal is fault management through load balancing in distributed SDN control plane. Architecture of the distributed SDN controller is briefly discussed here.

Distributed SDN controller share their network information in the SDN controllers. A worldwide network wide state is a product of aggregation from multiple network states. SDN controllers store the worldwide network state in the data store. SDN controllers can use any data store system whether it is relational database, non-relational database, distributed hash table, or distributed file system [7]. SDN adopter can use any schema within their data store. A controller can dominant control the network because it has a perspective of the whole network. Thus to provide a logically dominant control in distributed SDN controller, there must be at least one controller from the group that has a worldwide network state. Our survey summarizes following design choice.

(i) **Hierarchical model:** As shown in Fig. 2, one or few SDN controllers in the group have the worldwide network state. Two local controllers maintain the switch of each domain. Meanwhile the root controller accomplishes the management between local controllers. The worldwide network state which is kept in the data store is only available to the root controller. Thus the resident controller should initially question arrange data from the root controller before it can execute any inter domain activity. Because of this procedure this model is also called as client-server model, where the root controller acts as a server and local controller acts as a client. OpenDayLight [4], ONOS [5], Onix [8], and HyperFlow [9] are flat SDN distributed controllers.

(ii) **Flat model:** It is also called as horizontal architecture. As appeared in Fig. 3, three SDN controllers control switches in their space and every one of them have an
information store and keep up worldwide system state. The entirety of the SDN controller additionally own east/west bound API association with different controllers. Subsequently they can contact and tell other straightforwardly.

Before a SDN controller can build worldwide system state, it should initially get the neighborhood state from every SDN controller in the group. Additionally any progressions occurs inside the space of a controller must be shared to other controller with the goal that they can refresh their worldwide system state to mirror the changes. At the point when the bunch will have the option to play out each system data dispersion effectively, all the SDN controller ought to have the equivalent worldwide system state. Because of this procedure, administrators called this model as shared model since each controller can reach legitimately to different controllers during system data circulation. Others may called this model as repeated state machine on the grounds that the bunch duplicates all nearby system states from SDN controller to each other. Our model followed flat model architecture.

2.2 Related Work Done on Coordinator Election Algorithms in Distributed Computing Environment

Consistency and state replication in distributed SDN is represented by Zhang et al. [6] through Raft in SDN. Complicated interdependency introduced between network OS and network under its control on introducing consensus protocol for maintaining a consistent network state. Which generates new failures and instabilities. The connectivity among these controllers can be provided either in-band or out of band via data plane under their control. In either case (dedicated or virtual) it refers to the network connecting the controllers as the control network. Openflow switches with flow rules installed by the same controller cluster to which it provides connectivity in the network.

Esteban Hernandez [10] has discussed the performance of the OpenDayLight controller under the cluster architecture schema. The author has discussed the protocol of the network in the communication between controllers, different message exchanges between controllers. Message communication is done through the RPC mechanism, the authors proposed the coordinator election algorithm by “request vote” and “append entries”. To get the values from the other nodes request vote is used and to replicate log entries, append entries are used by the coordinator. This request vote is in the form of RPC messages. When this RPC message is blank, it is called “heartbeat” messages. The coordinator election process is executed for the arbitrary duration called “term”. A term always starts with an election process, where one or more nodes in the candidate state try to become a coordinator. When one of the candidates wins the election and became a coordinator, the term is conserved until it fails. There are also some situations, in which during a term there is not coordinator election, this can be caused by a split vote’s situation. In such a situation term will be completed without any coordinator, then a new election taking place with a new term. This ensures that only a node can become a coordinator within a given term. When nodes are started up, they begin in the follower state. They remain in that state as long as they do not receive any message from
a coordinator or candidate. If they don’t receive any message over a specified interval called “Election Timeout”. Then they will change their status to a candidate, assuming that there is no current coordinator.

Authors of [11] have utilized Raft consensus. A raft is a consensus algorithm for managing repeated logs. The consensus is an algorithm that enables controllers to work together as a unique coherent system that is able to handle the failure of some of the controllers. It can be done by replicating the state of the machine of the coordinator. Raft algorithm is divided into coordinator election, log replication, and safety.

Ricardo Mxacedo et al. [12] had represented algorithm based on controller performance. A controller having maximum performance is elected as coordinator. For that, at every regular interval, there is a need to measure the performance of each controller to find the best coordinator which can degrade the performance by overhead.

Scott D. Stoller [13] has used a modified Bully algorithm for coordinator election in which a separate failure detection module is required. We propose load calculation and decision making of the load balancing based on the controller threshold value.

Mazzini et al. [14] proposed dependability of SDN with distributed controllers over leader election algorithm and colored petrinet technique. The master controller will calculate the dependability rate of each node (controllers, switches) and link of the system. This calculation will be based on the proposed RDSDN [15] by the same author. The dependability of the system R (G) will be calculated as the possibility that every pair of the node can send and receive the data. Dijkstra algorithm will be used to calculate the link loss rate. Controller with highest reliability rate among all controllers and their children will be considered as coordinator controller.

We propose a coordinator election based on the load of the controller. All the controllers including the coordinator controller will calculate its own load using separate functions available with each controller, and send their load information to the coordinator controller at regular intervals. The load of the controller will be calculated by accumulating average message arrival rate, flow table entries, and propagation delay.

3 Objective of the Proposed Work

SDN is still in its maturing state so it suffers from issues such as scalability, reliability, and fault tolerance. Following objectives are derived from the reviewed literature:

- To find a suitable coordinator controller election algorithm to balance the load of the distributed SDN control plane.
- Provide more reliability, scalability, transparency and immediate deployability to distributed SDN controller using coordinator controller election algorithm.
- Verify claimed throughput (load balancing rate), latency (packet delay) with simulation results for with coordinator and without coordinator case.

4 Design and Implementation

Our primary research goal is to provide fault tolerance through load balancing in the distributed SDN controller. We have created a model called the DCFT (Distributed
Controller Fault Tolerance) model [1]. Figure 4 has described all the modules used in our model.

A sub-layer called fault tolerance layer, in SDN stack as shown in Fig. 4 is proposed in our model between application (management) plane and control plane by extending the application (management) layer. This layer holds different modules such as switch migration module, DCFT module, fault tolerance module, and transaction management module. The rest of the modules of the DCFT model are coordinator controller election module, Inter controller synchronization, load calculation and decision making modules are resting at the control plane. Out of all these modules, the coordinator controller election module, load collection & decision-taking module are focused on this paper.

The data plane contains forwarding switches as shown in Fig. 4. The control plane did the routing task based on the policy decision of the management plane. Each controller may have no switches with it. Openflow protocol will manage internal communication between controller and switches. Controllers are having three roles master, slave and equal [16]. Openflow protocols 1.5.1 specification [2] included the capacity for a controller to set its part in the multi-controller condition.

DCFT module is the main module of the proposed system, it stores the 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 the state of the controller, the output will be informing the user program about fault management.

Switch migration module provides load balancing and avoids failure on overloading. The overloaded switch will be migrated to a least loaded controller, selected from arraylist, maintained in store. The detail of this module is not presented in this paper due to a lack of space.

Control plane did routing task based on the policy decision of management plane, a load of each controller calculated, coordinator controller will be decided, inter controller messenger will publish update/sync controllers.

Different modules are described below.

**Application plane:** Store user program, perform event/command ordering, command execution, sharing distributed log records, zookeeper coordinating service is 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/sync update of the state of each controller using zookeeper

4 i/p: overloaded controller from the load calculation module

4 o/p: select the least loaded controller from the arraylist maintained in a distributed data store

4 o/p: select least loaded controller from arraylist maintained in-store, failure of any controller of the cluster due to any reason, orphan switches need to migrated to least loaded controller selected from array list maintained in the distributed log

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 (OCC) and distributed log

6(a) i/p: coordinator (master) controller failed before duplicating accepted events in the distributed log

6(a) o/p: call transaction management module, slave controller accept 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 duplicating the event but before commit request

6(b) o/p: the 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 requests

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 accept the confirmation that the switch processed the respective commands for that event and will not resend them (guaranteeing exactly-once semantics for commands)

7 i/p: It will receive updates from user programs about the state of the controllers. DCFT module saves the current state of the system, it will also reflect the changes/updates done by switch migration/fault tolerance/transaction management module. The inter controller messenger module provides coordination services through zookeepers via the DCFT module. While fault occurs, it will tolerate fault through fault tolerance and transaction management module

7 o/p: It will inform user programs runs on application plane about occurrence and management of fault

(continued)
Application plane: Store user program, perform event/command ordering, command execution, sharing distributed log records, zookeeper coordinating service is installed at application plane

Control plane:
2(a) i/p: Distributed SDN controller from the cluster
2(a) o/p: coordinator controller decided
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, the next coordinator will be elected from priority arraylist of coordinator maintained in a distributed log
1(a) i/p N- no of flow table entries, F-average message arrival rate, D-Propagation delay
1(a) o/p \( C_{\text{load}} = w_1^*N + w_2^*F + w_3^*D \)

Data plane: Forwarding packets as per the routing instructions received by the controller

4.1 Load Calculation and Decision Making Module

Proper load balancing gives better fault tolerance, better throughput of the system. Our model gives better load balancing rate, reduce packet delay. In the load calculation module, all the controllers including the coordinator controller calculate its own load and send load information to the coordinator controller. The current packet_in_rate of a controller is considered as a load of a controller. The 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 the array list is the minimum loaded controller and the last member is a maximum loaded controller without any duplicate entry. After a specified time-interval of every 5 s, the load calculation module calculates the load and sends it to the Coordinator. The time interval can be adaptive or dynamic. The time interval can be set by the aggregate of the current load and the previously calculated load.

4.1.1 Load Calculation Threshold

\[ T = \frac{T_{\text{max}}}{(|\text{CurrentLoad} - \text{PreviousLoad}| + 1)} \]

\( T_{\text{max}} \) = initially set interval

\( \text{CurrentLoad} \) = Controller’s Current Load

\( \text{PreviousLoad} \) = Controller’s Previous Load

Load of Controller = packet arrival rate of Controller (packets/Sec) \[2\].

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

4.1.2 Decision-Making Process

To balance the load of all the controller nodes, a threshold value \( C \) is decided to detect overload and under load conditions. Based on this threshold value coordinator decide to balance the load or not.
\[ C = \frac{\text{Average of a load of all the controllers}}{\text{a load of a maximum loaded controller}} \]

\[ 0 \leq C \leq 1, \text{ C is the load balancing rate. If C will be close to 1 load is evenly distributed and if a load is close to 0 uneven load distribution is there. We have selected an initial load balancing rate is 0.7. If the value of C is less than 0.7 then load balancing is required. If the value of C is greater than 0.7 no need for load balancing [8]. In case of overloading few switches need to be migrated from an overloaded controller to an underloaded controller, which is decided by the coordinator controller.} \]

4.1.3 Selection of Destination Controller and Switch to Be Migrated

After the decision taken by a coordinator to migrate a switch the next work to do is to select a destination controller to migrate a switch and selection of a switch to migrate. The selection of a destination controller can be from a sorted array list stored at the distributed data store. The lightest loaded controller has selected whose load is less than the below capacity CT. The selection of switch to migrate can be based on a packet in the rate of a switch. The maximum loaded switch should be select to migrate. Before the migration coordinator must check that the migrated switch should not overload the destination controller. It can be checked by the following formula. If the migration can create overload to destination coordinator should choose another switch to migrate.

\[ \text{Load of Switch to Migrate} \leq CT - \text{Load of Target} \]

\[ CT = \text{Controller Capacity (packets/Sec).} \]

4.2 Internal Controller Messenger Module

There are two different ways to obtain organize data from other SDN controllers. Polling and, Publish and Subscribe [12].

(i) **Polling**: Each SDN controller intermittently demand for another system data from different controllers in the bunch. For example a SDN demand for another switch data at regular intervals. It will execute the solicitation intermittently in any event, when there is no update occur in the other controller. In this manner it might get same system data as the last one. Along these lines this strategy isn’t effective [16].

(ii) **Publish and subscribe**

Each SDN controller can publish/subscribe in the system data from other controller in the group. For example controller c1 needs arrange data from neighboring controller c2, the controller c1 can subscribe in the switch data from c2. Right now goes about as c1 acts as subscriber and c2 goes about as publisher. Later c2 will advise c1 when there is a change with respect to the switch data in the area. Right now c2 will inform c1 just when there is a change accordingly this technique is increasingly proficient for our model [16].
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. ZMQ, the asynchronous messaging service used for internal communication among controllers. Distributed coordination service such as zookeeper [17] glues cluster of the controllers to share the information about a link, topology, etc. it’s used for updating the status of the controllers.

4.3 Coordinator Controller Election Module

Coordinator controller performs two roles, one is its ordinary role of routing incoming packets and second is special role, coordinator role, where it has to collect 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 their load information and switch information to the coordinator controller. The coordinator controller calculates the aggregate load of all the controllers and stores it in the distributed database. Based on a load of the cluster, the coordinator controller takes the switch migration decision. Controllers can communicate with coordinator using messaging services provided by zookeeper and sync service of floodlight controller.

This module is providing fault tolerance to the coordinator controller. There must be a coordinator controller all the time available 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. 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 otherwise it sets the controller having id one as the default coordinator.

Controllers of a distributed control plane form a logical cluster. Consider the controllers C1, C2, C3… Cn from Fig. 4, all the controllers are joined with switches in three different roles, master, equal and slave. C1 joined with S1 as master, S2 as a slave, C2 joined with S2 as master, S3 as a slave, similarly controller are joined with a master-slave relationship in the different switches of the clusters.

All controllers of a cluster are assigned a controller id as per they joined the controller cluster viz. C1, C2…Cn. When cluster start, the controller having maximum controller id is elected as a coordinator controller using our election algorithm. Proposed coordinator controller election algorithm
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 via publishing and subscribing updates across all nodes
3 pollTime=5 seconds, timeout=6 seconds
4. First, try to get a 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 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 elections if the majority of a 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 an entire network, none or multiple coordinators causes it to set the current state to ELECT.
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 >= majority (51%)) 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 the election: It follows the current coordinator
29. Nodes joined before the election: It participates in the current election process, a 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 participate in the election, and starts following elected coordinator after the election.

(i) 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
Coordinator Controller Election Algorithm to Provide Failsafe

Failure in coordinator controller, the election of new coordinator controller to failure of a whole distributed control plane. Failure of coordinator can be detected by using separate functions available with all the controllers in the cluster which will be synchronized with zookeeper services. 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 the elected coordinator is active or failed. If the coordinator failed, the re-election starts. A controller having maximum controller id from the cluster, elected as a new coordinator of a distributed control plane. A new coordinator has to migrate switches of the failed controller to a lightest loaded controller by proposed switch migration. All the controllers may have a different number of switches. Figure 5 shows failure in the 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.

(ii) Failure in coordinator controller

All the controllers are mapped with the no of switches. The controller is responsible for the flow table stored at switch mapped with that controller. If the controller gets failed, we have to migrate its switches to another controller to keep switches functioning.

(iii) Load imbalance between controllers

Similarly, load imbalance occurs on the 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.
5 Evaluation and Results

We have created a custom topology named mytopo.py. Here in our topology four floodlight controllers included. Each having two switches with a master role. There is only one master controller and many slave controllers in the topology. The controller may have more than one switch. There are many slave controllers for the switches. If the original master fails, a slave controller will be chosen as a new master. We consider custom topology in Fig. 4. Traffic patterns are shown in Table 1 used for all simulations. For example Controller c0 is connected with switches s1 and s2 with the master role and with switches s3, s4, s5, s6, s7 and s8 with slave role. A red dotted line indicating the master connection of the controller with its switch. To minimize the complexity here we have not displayed slave connections of all the controllers and switches. The topology is created using miniedit, a graphical user interface of Mininet.

5.1 Implementation of Coordinator Election Module

This module provides fault tolerance to the control plane when the current coordinator gets failed and at the same multiple instances of the controller are running. It uses ISyncService in order to store the data. Few updates published by all other modules in the controllers this module synchronizes state between the controllers, In addition, it runs a coordinator election process, in order to enable modules to perform role-based programming in a distributed system with multiple controllers running, in addition, be able to communicate between controllers as well.

In case of failure of the current coordinator there are the following cases:

Case 1: Majority of the controllers are active:
A normal coordinator election takes place and a new coordinator pops up, and the system operates normally.

Case 2: Less than 51% of configured controllers are active:
In this case, the system by default, elect controller 1 as a new coordinator. This is because the coordinator must be available all the time to perform its role.
**Case 3:** Node joins before the election: It participates in the election, the coordinator is elected amongst the currently active and configured controllers.

**Case 4:** Node joins during the election: It waits for election to be completed does not participate in the election, following the elected coordinator.

### 5.2 Experimental Setup

We have used Ubuntu 14.04 with 4 GB RAM, Intel Core i3-2370 M CPU@ 2.40 GHz processor system. The bandwidth of the system is 1 GBps. In four terminal windows, four floodlight controllers are started with similar IP address along with port no 4242, 4243, 4244, and 4245. The default Coordinator will be ControllerID1 but after starting 3 controllers, packet arrival rate (throughput) will be tested by changing the number of the controller node. The throughput and latency mode of the cbench will be used in the floodlight controller to check the throughput (packet arrival rate), latency, communication overhead of the floodlight controller.

TCP flows generation needed to simulate the distribution of network traffic the average flow requests. It's done by hping3. The average packet arrival rate of 500 packets/s. Floodlight controller will be used to process packets received by the switch. To reduce the effect of packet delay and packet loss link bandwidth between switches and hosts to 1000 Mbps. 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 h readings noted at every 20 min.

### Table 1. Traffic design used in the experiment

| Traffic sequence | Source | Destination |
|------------------|--------|-------------|
| T1               | H1     | H4          |
| T2               | H3     | H7          |
| T3               | H1     | H8          |

### 5.2.1 Paket Delay (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. The simulation experiment starts with a packet delay of 12–14 ms for all traffics. Initially, there is no coordinator, so by default C1 will be chosen as coordinator, as all the four controllers joined with topology and if C1 failed, the coordinator election algorithm will be executed. Figure 9 reveals that packet delay will be decreased in T1 and T2 with the coordinator. Because the proposed algorithm followed the state replication approach. All the controller followed the log replication of the coordinator. The average packet delay of T1, T2 and T3 traffic sequences is decreased by 56.13%.
Table 2. Comparison of with coordinator/without coordinator controller in the cluster

| Traffic number | Packet delay (ms) | Communication overhead (packet/s) | Throughput (packet/s) | Packet delay (ms) | Communication overhead (packet/s) | Throughput (packet/s) |
|----------------|-------------------|-----------------------------------|-----------------------|-------------------|-----------------------------------|-----------------------|
|                | Switch-controller | Controller-controller | C1 | C2 | C3       | Switch-controller | Controller-controller | C1 | C2 | C3 |
| T1             | 58.1              | 403                              | 352 | 423 | 421 | 379       | 28.66       | 413 | 362 | 543 | 541 | 499 |
| T2             | 68.9              | 374                              | 301 | 398 | 444 | 354       | 26.74       | 394 | 356 | 518 | 564 | 474 |
| T3             | 55.7              | 234                              | 202 | 354 | 424 | 388       | 24.20       | 267 | 264 | 514 | 544 | 508 |

5.2.2 Throughput

Figure 7 shows the comparison of the cluster throughput (packet/s) with and without the coordinator controller. Packet_in_rate is considered as the throughput of the system. Table 2 and Fig. 5 reveals that with respect to the topology shown in Fig. 6, packets are captured through Wireshark and analyzed from all the traffic sequences like T1, T2, and T3 at different times. Average throughput will be increased by 22.63%.

5.2.3 Communication Overhead (KB/S)

Figure 8 shows the comparison of the communication delay (KB/s) with and without the coordinator controller. It is calculated between switch-controller and controller-controller for all the traffic sequences T1, T2, and T3. With the coordinator controller, communication overhead between switch-controller will be increased by 7.09 KB/s. and between controller-controller, it is increased by 23.47 KB/s. In the availability of the coordinator controller, the replication state machine approach followed, so log replication carried out at regular intervals resulted increase in communication overhead.
6 Conclusion

This paper 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. The coordinator controller election algorithm, load calculation, and decision-making modules are described in detail. There are four floodlight controllers in the sample topology (Fig. 4), taken for the simulation. The result of the simulation will be tested for three different traffic sequences as shown in Table 1. Comparison of with coordinator controller and without coordinator controller is demonstrated by Table 2. 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 and immediate deployability of the system are improved at the cost of communication overhead.

Acknowledgements and Conflict of Interest. We are very much thankful of Dr. Bhushan Trivedi, Director, GLS Institute of Computer application, Ahmedabad, Gujarat, India, Dr. Satyen Parikh, Dean, Faculty of Computer Application, Ganpat University, Kherva, Mehsana, India, all our colleagues, friends and reviewers, and organizer of the International conference on computing science, communication and security (COMS2)-2020 the for their comments on our research work and help us to improve this paper. We are hereby declare that we don’t have any conflict of interest with any living person about the said research work mentioned in this paper.

References

1. Lakhani, G., Kothari, A.: Distributed Controller Fault Tolerance Model (DCFT) using load balancing in software defined networking. Int. J. Comput. Eng. Technol. 10(2), 215–233 (2019)
2. “OpenFlow switch specification 1.5.1”, p. 74. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. Accessed 13 Feb 2019
3. Jarraya, Y., Madi, T., Debbabi, M.: A survey and a layered taxonomy of software-defined networking. IEEE Commun. Surv. Tutor. 16(4), 1955–1980 (2014)
4. Suh, D., Jang, S., Han, S., Pack, S., Kim, T., Kwak, J.: On performance of OpenDayLight clustering. In: IEEE NetSoft Conference and Workshops (NetSoft), pp. 407–410 (2016)
5. Berde, P., et al.: ONOS: towards an open, distributed SDN OS. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, pp. 1–6. August 2014
6. Zhang, Y., Ramadan, E., Mekky, H., Zhang, Z.L.: When raft meets SDN: how to elect a leader and reach consensus in an unruly network. In: Proceedings of the First Asia-Pacific Workshop on Networking, pp. 1–7, August 2017
7. Oktian, Y.E., Lee, S., Lee, H., Lam, J.: Distributed SDN controller system: a survey on design choice. Comput. Netw. 121, 100–111 (2017)
8. Koponen, T., et al.: Onix: a distributed control platform for large-scale production networks. In: OSDI, vol. 10, pp. 1–6, October 2010
9. Tootoonchian, A., Ganjali, Y.: Hyperflow: a distributed control plane for OpenFlow. In: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking (2010)
10. Esteban, H.M.: Implementation and performance of an SDN cluster-controller based on the OpenDayLight framework (2016)
11. Suh, D., Seokwon, J., Sol, H., Sangheon, P., Taehong, K., Jiyoung, K.: On performance of OpenDayLight clustering. In: 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp. 407–410 (2016)
12. Macedo, R., de Castro, R., Santos, A., Ghamri-Doudane, Y., Nogueira, M.: Self-organized SDN controller cluster conformations against DDoS attacks effects. In: Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2016)
13. Stoller, S.D.: Coordinator election in distributed systems with crash failures. Technical report, Indiana University, p. 169, April 1997
14. Mazzini, S., Mohammad, Reza K., Naser, M.: Improving the reliability of software-defined networks with distributed controllers through leader election algorithm and colored petri-net. Wirel. Personal Commun. 109(1), 645–656 (2019)
15. Moazzeni, S., Khayyambashi, M.R., Movahhedinia, N., Callegati, F.: On reliability improvement of software-defined networks. Comput. Netw. 133, 195–211 (2018)
16. Zhou, Y., et al.: A load balancing strategy of SDN controller based on distributed decision. In: 2014 IEEE 13th International Conference on Trust, Security, and Privacy in Computing and Communications (TrustCom), pp. 851–856. IEEE (2014)
17. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: ZooKeeper: Wait-free Coordination for Internet-scale Systems. In: USENIX Annual Technical Conference, vol. 8, no. 9, June 2010