Wireless Personal Communications (2020) 114:3507–3539
https://doi.org/10.1007/s11277-020-07545-2

Fault Administration by Load Balancing in Distributed SDN Controller: A Review

Gaurang Lakhani[1] · **Amit Kothari**[2]

Published online: 10 June 2020

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 (SDN) brings all the functionalities to a single location and making centralized decision. Controllers are the main entity 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. Through this paper, we presented basics of distributed SDN controllers with its properties, studied classification of SDN controllers vide their logical design (hierarchical/flat model), programming language, adaptability, application domain etc. We categorize fault management issues in management plane, control plane and infrastructure plane and their interface. Paper will be concentrated on the faults generated through overloading of the controllers. As a solution switch migration technique will be derived. Provide detail discussion on prior work done in switch migration techniques for fault management through load balancing in distributed controllers. At last addition of fault management plane will be proposed in the SDN stack for generating robust distributed SDN controller.

Keywords Software defined networking · Fault management · Resiliency · Fault detection · Fault prevention · Fault recovery · Failure · Survivability

1 Introduction

SDN separated control sense from its underlying hardware and its centralism is software based controllers. In Multiple controller environment they are physically distributed, logically centralized. It generates issues like scalability, reliability, availability, consistency, security etc.

✉ Gaurang Lakhani
gvlakhani1@gmail.com

Amit Kothari
amitdkothari@gmail.com

[1] Gujarat Technological University, Ahmedabad, Gujarat, India

[2] Accenture, Pune, India

SDN structure shadows top-down coherently centralized network control, SDN controller with SDN application installed can handled all the switches in the network. By this midway control network handled proficiently and respond the dynamic incident spontaneously.

This logical centralization managed by single and distributed SDN controller. All the switches are handled by one SDN controller in single SDN controller. But scalability and robustness issues arise in single SDN controller.

1.1 Scalability Issue

In general, scalability refers to the ability of the network to scale and control high amount of traffic, In SDN, the scalability reflects the capacity of SDN controller in handling multiple path forwarding requests from switches. SDN controller has inadequate sources when conduct high amount of requests. To resolve the problem many investigators bound the forwarding path request sent to SDN controller [1–3]. It will intelligence to the switches and violating the concept of SDN.

1.2 Robustness Issue

Single point failure in single SDN controller resulted failure of the network. All the switches will be failed and cannot forward new packets. Eventually whole network will be down. Investigator concentrate on openflow-hybrid [4] to convert openflow switch to traditional switch, when it fails to connect to the SDN controller.

Figure 1 exhibit example of several SDN controllers works individually in routing packets through several domains. The Packet-In message is sent to ask the forwarding path from the SDN controller. The Flow-Mod message is forwarded to install the forwarding path to the switches.

Distributed SDN controller helps to solve above issues. In general multiple controllers are used to share the workload of the network. On crash of the controller, another controller take possession. Few concepts from the distributed system are adopted for it.

Figure 1 shows two different SDN controllers. Each of them manages a network part called network domain. c1 manages switches of domain1, similarly c2 manages switches of domain2. Now suppose h1 wants to send few packets to h2. As packets reaches to s1, s1 desires for a forwarding path from c1 via openflow packet-in message. Once forwarding path received, it will send the packet to s2. Similarly when packet reaches at s3, s3 wants the promoting path from c2 and eventually packet arrived at h2. Here two SDN

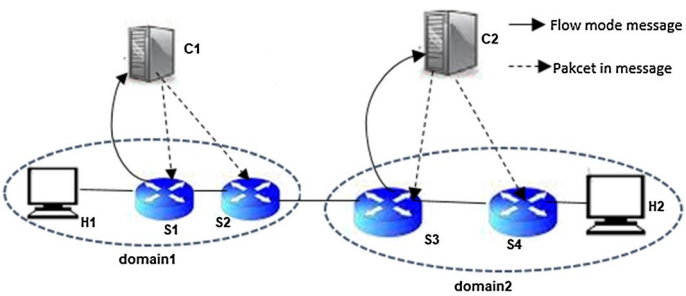

Fig. 1 Multiple SDN controller

controllers are used but they work alone for each network domain and does not representing distributed SDN controller.

If c1 and c2 both sharing same logic, new packet arrived at s1, both c1 and c2 directly install forwarding path to all the corresponding switches. Then it is called distributed SDN controller.

Properties of distributed SDN controller such as network domain, local (static and lively network state), global network state and connection between controller-controller as East/Westbound API are mentioned in [5]. Researchers of the same paper explored design choice of distributed SDN controller including (i) switch to controller connection strategy as IP alias connection or Master/Slave connection, (ii) network information distribution strategy vide hierarchical or flat model, (iii) controller connection strategies in distributed SDN controller as coordinator based and coordinator less, (iv) in-band vs. out-of-band connection strategies.

This paper presented Survey on distributed SDN controller with special focus on fault tolerance in distributed SDN controller. Paper studied basics and properties of distributed SDN controller, Comparison of centralized and distributed control plane architecture studied from previous research.

Few previous surveys conducted by network administrators show that the normally errors in network failures can be generated due to logical error of switch/router programming, wrong configuration of protocol, hardware failures, and external factors. Common indication of failures are reachability problems, tained congestion and latency/throughput [5]. Different kinds of faults occurred at various layers and layers/interface in distributed SDN controllers are studied in the paper. Finally fault management through load balancing with switch migration techniques are surveyed.

Main contribution of this paper are as follows:

- Presenting the review of classification of SDN controllers.
- Providing a complete systematic review of different types of fault issues in distributed controller at application, data and control planes and into their interfaces.
- Studied existing techniques switch migration issue in load balancing among distributed controllers.
- Proposed deployment of fault tolerance plane in the SDN stack. Using such additional fault tolerance plane, derived a novel load balancing model, which provides robust distributed SDN controller.

Rest of the paper contains Sect. 2 studies all the existing SDN controllers and tabularized comparison between their documentation, programming language, adaptability, application domain, their Northbound/Southbound API etc. Section 3 discussed about different types of faults in the distributed SDN controller. Section 4 concentrated on the faults occurred through overload and as a solution applied switch migration technique.

Different existing strategies studied for switch migration and the survey concluded with the decision that for efficient fault management through load balancing a separate fault management layer need to be derived from the application layer in the SDN stack. The fault management layer includes different modules viz as switch migration module, module maintaining state of the network, fault management, transaction management for the transactional updates of the network. So for survey ends with the aim of developing a novel model for robust distributed SDN controller which provides, consistency, reliability and scalability.

2 Classification of SDN Controllers

SDN controllers can be centralized controllers or distributed controller. Table 1 shows comparison of distributed and centralized controllers with respect to documentation, adaptability, OS, Northbound, southbound API and application domain. SDN controller architecture divided into logically or physically centralized and flat, distributed in hierarchical or hybrid design. Table 1 listed few controllers among it.

Table 2 shows fault tolerance comparison of different distributed controllers. Controller failure/link failure are major sources of fault. Their solutions like nearby backup controller, backup paths,active replication, dynamic controller migration, forwarding policy reconfiguration, replicated shared database for recovery are discussed.

Table 3 demonstrate mechanism of fault tolerance such as controller replication, failure recovery, protection switching, segment protection, fast failover, packet modification.

3 Fault Management Issues in Distributed Controllers

Fault management process include fault detection, fault localization and fault recovery. Fault prevention can be done proactive or reactively. Distributed controllers in SDN contains infrastructure (data) plane, control plane and application planes. Control plane consists of number of controllers, all controllers are attached with east–west bound interfaces. Similarly application plane/control plane are connected with northbound interface and control plane/application plane are connected with southbound interfaces. We discussed different types of faults in distributed controllers based on their layers and interfaces. Figure 2 summarizes different types of faults. Different protocols such as OSPF [47], AMQP [50] etc. are used for solving problems generated by different types of faults. Prior work listed fault management in SDN network [51–55]. They have not concentrated specifically in distributed controller. We concentrate on fault management at control plane in distributed controller. Control plane consists its architecture, controller location methodologies, and traffic fault tolerance. All this category encompasses multiple issues leads to faults such as control channel disruption, controller disappointment. Kreutz et al. [52, 53] depicted about fault tolerance and reliability issues. Silva et al. [53] described resilience discipline while Sharma et al. [55] described efforts related with fault detection and fault recovery.

Fault recovery divided into protection (pro-active) and restoration (re-active) method. In protection backup paths are preconfigured using fixing of flow rules in the switches. Restoration follows notification strategy. On failure, notification triggered to controller, controller examines the notification and decide the type/level of fault. It calculates alternative path and installing flow of switches in case of link failure. Restoration and protection both method have its merit/demerits. Normally they are used to reduced recovery time, bandwidth optimization, and minimum TCAM consumption.

Classification with respect to layers and layer/interface in distributed controller shown in Fig. 2.

3.1 Data Plane (Infrastructure Layer)

Fault tolerance issues such as link failure and node failure, normally available in traditional network are part of this layer. However in SDN due to centralized management and

Table 1 Classification of SDN controllers [24]

Sr no.	SDN controller	Programming language	Documentation	Adaptability	Distributed/ centralized	OS	Southbound APIs	NorthBound APIs	Application domain
1	ONOS [6]	Java	Good	High	D	Linux, MAC, windows	OF1.0,1.3,NETCONF	REST API	Data center, WAN, Transport
2	ONIX [7]	C++, Python, Java	Good	High	D	Linux	OF 1.0,1.3,1.4	—	Data center
3	Hyperflow [8]	C++	Good	High	D	Linux	OF 1.3,1.4		Data center/WAN
4	OpenDayLight [9, 10]	Java	Very Good	High	D	Linux, MAC, windows	OVSDBOF1.0,1.3, NETCONF, Yang, LISP BGP/LS,SNMP, PCEP	REST API	Data center
5	Runos [11]	C++	Fair	Fair	D	Linux	OF 1.3	REST API	WAN, Telecom, Datacenter
6	Big Cloud Fabric [13]	Java	Good	Fair	D	Linux	OF 1.3,1.4	vSphere, NSX, SAN	Data center
7	BORON [12, 14]	Java	Good	High	D	Windows, Linux, MAC	NETCONF, Yang, OVSDB, OF1.0,1.3, PCEP, BGP/LS, SNMP,LISP,	REST API	Data center
8	OSC [15]	Java	Good	High	D	Linux, MAC, windows	OF1.0,1.3,NETCONF,	REST API	Data center
9	DISCO [16]	Java	Good	High	D	Linux	OF 1.3.,1.4	AMQP [109]	Data center/WAN
10	Kandoo [17]	C, C++, Python	Good	Fair	D	Linux	OF 1.0,1.2	RPC	Data center
11	Fleet [18]	Java	Good	High	D	Linux	OF 1.3,1.4	Adhoc API	Data center
12	PANE [19]	Java	Good	Fair	D	Linux	OF 1.3, 1.4	REST API	Data center
13	Ravana [20]	Python	Good	High	D	Linux	OF 1.3,1.4	RSM	WAN/Data center
14	Meridian [21]	Java	Good	High	D	Linux	OF 1.3, 1.4	REST API	Data center
15	NOX [22]	C++	Poor	Low	C	Linux	OF 1.0	REST API	Campus
16	POX [23]	Python	Poor	Low	C	MAC, windows, Linux	OF 1.0	REST API	Campus

Table 1 (continued)

Sr no.	SDN controller	Programming language	Documentation	Adaptability	Distributed/ centralized	OS	Southbound APIs	NorthBound APIs	Application domain
17	RyU [24]	Python	Fair	Fair	C	Linux	OF1.0,1.2,1.3,1.4,NET CONF, OF CONFIG	REST for South-bound	Campus
18	Beacon [25]	Java	Fair	Fair	C	Linux	OF 1.0	REST API	Research
19	Maestro [26]	Java	Poor	Fair	C	Linux	OF 1.0	RISE,NSF	Research
20	Floodlight [27]	Java	Good	Fair	C	Linux, MAC, Windows	OF 1.0,OF 1.3	REST API	Campus
21	IRIS [28]	Java	Good	Fair	C	Linux, MAC, Windows	OF 1.0, OF 1.3,OVSDB	REST API	Carrier-grade
22	MUL [29]	C	Fair	Fair	C	Linux	OF1.3,1.4,OVSDB, OF CONFIG	REST API	Datacenter
23	NOX-MT [30]	C++	Poor	Low	C	Linux	OF 1.0	REST API	Campus
24	HP VAN [31]	HP-Proprietary	Fair	Fair	C	Linux	OF 1.0	REST API	Campus
25	VM-ware NSX [32]	C++	Good	Fair	C	Linux	OF 1.0	REST API	Campus
26	Programmable Flow Controller (NEC) [33]	NEC-Proprietary	Good	Fair	C	Linux	OF 1.0,1.3,1.4	REST API	Campus
27	vneio/sdnc [34]	Java, Python-Ruby	Good	Fair	C	MAC, Linux	OVSDB	Intel DPDK Tech-nology	Data center
28	Cherry [35]	Go language	Fair	Fair	C	Linux	OF 1.0,1.3	Proxy ARP,L2 Switch, Float-ing IP	Data center
29	OpenContrail [36]	Python	Good	Fair	C	Linux	OF 1.3	REST API	Cloud/WAN
30	Faucet [37]	Python	Good	Fair	C	Linux	OF 1.3	Zodiac FX	Data center

Table 1 (continued)

Sr no.	SDN controller	Programming language	Documentation	Adaptability	Distributed/ centralized	OS	Southbound APIs	NorthBound APIs	Application domain
31	VSC [38]	Cybil by Nokia	Good	Fair	C	Linux based SR-OS	OF 1.3,OVSDB	REST API	Data center
32	VortiQa [39]	C	Good	Fair	C	Linux	OF 1.3	ON director	VPN/WAN
33	B4N [40]	Java	Good	Fair	C	Linux	OF 1.3,1.4	REST API, NET-CONF	Carrier Grade SDN
34	Orion [41]	C, C++, Python	Good	Fair	C	Linux	OF 1.0,1.2	RPC	Data center
35	B4 [42]	Python	Good	High	C	Linux	OF 1.0,1.3	BGP	WAN
36	Espressso [43]	Java	Good	High	C	Linux	OF 1.0,1.3	BGP	WAN
37	SmartLight [47]	Java	Good	Fair	C	Linux, MAC, Windows	OF 1.0,OF 1.3	REST API	Campus
38	SWAN [43]	Python	Good	Fair	C	Linux	OF1.0,1.3	BGP	Data center
39	Rosemary [44]	C, Python	Poor	Good	C	Linux	OF1.0,1.3	REST API	Data center

Table 2 Fault tolerance comparison of distributed controller platform

Sr no.	Controller platform	Failure type	Solution
1	Hyperflow [8]	Controller failure	Nearby controller serve as a standby controller
2	Onix [7]	Link/switch failure, Onix instance failure	Backup paths, active replication
3	ONOS [6]	ONOS instance failure	Redundant instances
4	Hydra [45]	Controller disappointment	Replication of controller failure
5	Elasticon [46]	Controller disappointment	Migration with dynamic controller
6	Fleet [16]	Link disappointment	Switching to work around path for the futile link
7	IRIS [28]	controller disappointment	Controller switching
8	PANE [17]	Link/switch disappointment	Forwarding policy regeneration
9	Smartlight [47]	Controller disappointment	Pretend collective database for retrieval

programmability leads to new challenges. Failure detected at link or node level. Sharma et al. [56] proposed solution of Loss of Signal (LoS) as BFD (Bidirectional Forwarding Detection) Loss of signal recognizes interface failure by checking whether a particular port of a switch is down.

3.1.1 Network Failure Discovery and Position

In legacy networks, in fault discovery and finding place, number of challenges raises: a single disappointment may leads to many source of errors, long time needed to stabilize the state of network devices. Even partition of network may be difficult to detect. Due to logical centralization, SDN keep all the devices of network updated about information of any fault related with link or node. Only failure affected switches will be informed by controllers. Rest part of the network may work in normal fashion. Extra care need to take to reduce the communication recovery time in distributed controller.

3.1.2 Network Failure Recovery

Protection and restoration are two approaches for network failure recovery. Preconfigured backup path followed in protection. On demand reinforcement path are generated in rebuilding or restoration. Improper design of backup path in protection, not genuinely circulated among the connections, may lead to blockage situation. If large forwarding rules in the switch it will be exhaust. Any other way disappointment renovation increases retrieval period, overburden network controllers etc. distributed controllers in SDN cares hybrid approach by numerous protocols and mechanisms for improving failure recovery.

3.2 Control/Infrastructure Interface

South bound protocols such as OpenFlow [4, 57], Forces [57], OpFlex [46] etc. are the major protocols used between control plane and information plane interface. SDN separates control logic from network devices and put it in centralized fashion in control plane of distributed controller. Data traffic and control traffic links connecting different planes of SDN. We discuss control network disappointment and controller-switch dependable

Table 3 Different types of faults and mechanism of fault tolerance [48, 49]

Sr. no	Mechanism of fault tolerance	Defect types	Features
1	Controller repetition	Controller defect	Uninterrupted communication between switches and controller
2	Disappointment recovery	Node or link defect	Switches retrieval without controllers retrieval time within 50 ms
3	Protection switching	Node or link defect	fault retrieval time within 50 ms
4	Segment defense	Node or link defect	Avoid controller role in retrieval process by 64 ms
5	Fast failsafe	Node or link defect	Explore network flexibility
6	Packet alteration	Node or link defect	Carry backup route to alter the forwarding policies

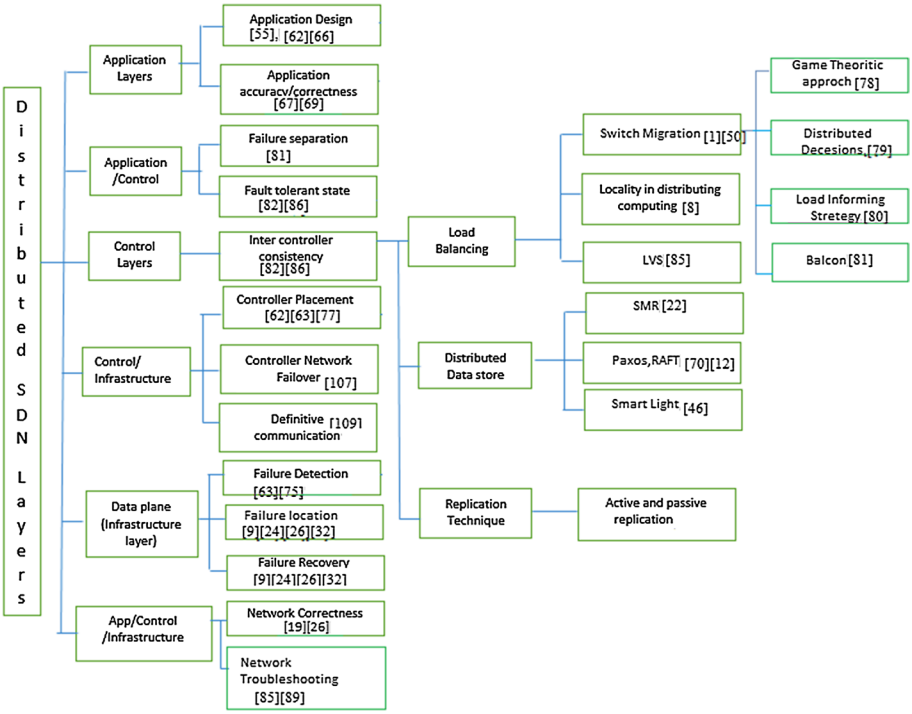

Fig. 2 Fault management issues in distributed controller [82]

communication issues which can be proportionate to link or node disappointment and data distribution in traditional network. Along with this in distributed controller, controller location and controller failover also discussed.

3.2.1 Control Network Failure

Distributed controller consists massive number of links, used for connecting all the controllers, and rest of the devices in the SDN. All these links manages network. Faulty link of the switch can be interrupted normal operation of devices associated with it and switches. If standby paths calculated using restoration or protection technique, control channel might prevent data traffic reclamation. Aim of the control traffic recovery is to reconstruct switch connectivity with any available controller. In case of data traffic retrieval specific destination path must be rebuild. Experiment of [58, 59] proves that well organized, effective control channel failure identification is a significant issue to decrease retrieval time.

3.2.2 Controller-Switch Dependable Communication

Separation logic of control plane and data plane in SDN leads to check the reliability of event execution. If event is not delivered to its endpoint, network management spoiled. For example control plane don't get information of link failure, routing services generates invalid paths as it does not know about failure of link. In data plane new device needed to be installed in the network, and not validated by authentication service, device cannot be

connected. Reliable message delivery is crucial problem, wide previous work and many protocols discussing this problem, In SDN also this problem is thoughtful since the messaging between centralized control and underlying architecture devices is very important and present itself in new forms e.g. network actions can be vanished during controller failover.

3.2.3 Controller Position

Distributed controller consists multiple controllers. Multiple controllers environment generates two questions in the designing of network (1) what number of controllers are expected to fulfill the network desires? (2) What is the position of the controller in the topology? Fewer number of controllers tends to overload on existing controllers and cause stoppage of service due to resource overburden, while access number of controllers may lead to underutilization, wasting of money and resources. Author in [60] shows position of the controller in the system configuration affect network execution and its output. As availability amongst controllers and rest of the devices of information plane is significant to arrange administration, a non-ideal controller position permitted in the event of just smaller of connection fails, the vast mainstream of the devices still drop control channel movement. Retrieval time and dormancy between inter controller influenced because of wrong position of controller.

3.2.4 Controller Failover

Leveled or nonhierarchical design followed by distributed controllers. In leveled mode switch guides all its demand to a primary controller and on its failure switch connect with a preconfigured reserved controller. On the other hand nonhierarchical mode where switch joined with numerous controllers simultaneously and sends each demand to every one of them, controllers are managing order of the requests and reactions. OpenFlow care this role of the demand from its version 1.2. All switch can cosign roles to the controllers connected to it: master have all reading/writing access to the switch, one controller can be master role only for a given point of time. Slave role only reading state enabled, cannot control the switch, writing in the flow table; and other role is equal, similar to master controller can manage the switch without any confinements and any number of controller can be built as equal. Controllers may have different in migration cost, flows/second, response time, latency etc. it leads to the question, which controller will be picked as first prime? How to manage sequence of backup controller? How regularly they are refreshed? How to detect failures among primary and secondary controllers? In nonhierarchical mode, failover process is modest. As all controllers are as of now associated, demands must be requested in such way that any of the controllers can react to any demand amid the failover.

3.3 Control Layer

Distributed controllers consists multiple controllers, control layer is intermediate translation layer among application and data plane. Inter-controller communication exclusively refers control layer own operations, rest of the operations are affected by the other layers.

3.3.1 Inter-Controller Consistency

Centralized or distributed controllers gives network services while increasing resiliency. Numerous controller used to maintain strategic distance from SPOF (single point failure) [6, 20]. For logical centralization all the controllers must rake-off worldwide network view.

Levin et al. [58] depicted state distribution trade-offs and perform tests particularly on consistency levels. Controller irregularity prompts to issue of state consistency among primary and reinforcement controller. False assumption made by controller and controller and network device should not be managed properly. Event delivery ordering must be imposed according to the order of controller consistency for dynamic replication.

3.3.2 Control Plane State Redundancy

Primary-backup relationship in multiple controllers followed by centralized controller. In centralized controller primary controller is managing whole domain, backup managing consistency of its state with primary. In distributed controllers, different controllers are interchange information among each other with synchronization. They can take control of the network simultaneously.

Control state redundancy achieved by three different ways vide state replication, event propagation and activity (traffic) duplication. State replication accomplished by replicating application state into replicas of controller, event propagation managing order of the events sent to all replicas for consistency. Activity duplication, duplicates all movement at switch level into copy controllers.

3.3.3 Distributed Data Store

Multiple controllers sharing network view for consistent network operation via distributed data store. In hierarchical or nonhierarchical mode network divided into partition. Each partition have its own master and more than one backup controller. View to the partition shared to other partition via distributed data store.

Onix [7] holds Network Information Base (NIB), where network view is represented in a graph. Onix is distributed controller. Network operations done by using NIB and distributing state of the network among controllers. Onix coordinates different storage strategies: DHT (Distributed hash table) for weak consistency and transactional storage for strong consistency. Onix supports Zookeeper [59] for leader election and failover.

ONOS [6] supports high scalability and high availability. Similar to Onix, it uses graph and store for each service state. ONOS uses cluster with multiple instance for distributed mode. Where each node is taking care for subset of network devices. Publish-subscribe mechanism used by ONOS for achieving different level of consistency. ONOS upgrading with different versions. It uses different distribution mechanism such as Cassandra [61] data store for ultimate consistency, Hazelcast [62] and zookeeper [59] for strong consistency.

HP Virtual Application Network (VAN) [29] is implemented through OSGi [63] specification, it provide application independence and reuse. AMQP [50] used by HP-VAN controller for message delivery at application layer. Similar to Onix, HP-VAN did coordination and synchronization with Zookeeper [59]. It offers clustering mode.

Spalla et al. [64] proposes concept which provides basic clustering capabilities. Master–slave relationship followed among multiple controllers in each partition of the cluster. State sharing of controllers done by OpenReplica [65]. It is coordinating service for consistency and synchronization. It also used Paxos consensus protocol [66].

3.3.4 Control Plane State-Update Messages

Passive repetition approach in integrated architecture is another approach for consistency. In this approach application state and state update messages are encapsulated together into state-update message and send it to backup controllers for updating their own state. Fonseca et al. [67] proposed CPRecovery, it is inert repetition mechanism. Inert repetition divided into replication and retrieval phase. In replication phase the primary controller generating backup replicas on the change of state. While replicas orchestrating their state with the primary. On detection of failure status from primary controller (link failure, heartbeat message) the retrieval phase started. Switch joins to the constituted standby controller, which apprises its role to primary, network did its works normally.

High availability controller (HAC) architecture, a crossbreed method that substitutes between both strategies, rendering to service precedence proposed by Pashkov et al. [68]. Network updates shared by controllers. Each application data updates also shared with all controllers. State of the controller stored in shared data store. Serialization helps to coordinate and ordering of message services.

3.3.5 Event Replication

Event oriented paradigm followed by openflow. All network changes transfer to the SDN controller for appropriate processing. Hence, it is conceivable to accomplish state repetition rerunning occasions in standby copy. Because of high activity of system occasions controller might be over-burden.

Hyperflow [8] provides scalability by spreading designated events. Events using publish-subscribe system for propagation. Hyperflow [8] will configure orphan switches to its nearest neighbor on failure of controller. It does not guarantee event ordering. DISCO [16] using AMQP [50] protocol for event propagation, guaranteeing message delivery. It offers bandwidth use and latency also.

Viewstamped replication [69] utilized by centralized controller Ravana [20] to imitate events and offer fault tolerance in information plane and control plane. repetition of controller state and switch consistency accomplished by two stage protocols which holds properties like aggregate event ordering, precisely onetime event processing and preciously onetime command execution. In total event ordering all controller replica follows the same order of events. Precisely onetime event processing the controller continuously process all the events issued together. Events lost due to any reason must be retransmitted in the same order. Exactly once command execution means switch executes commands once and only once.

Ravana architecture followed system like: a master controller receives request from the network, stored in shared replicated log and process it. The replica works on request only after processing completion of master. The replica controller keeps log record of the events by its IDs to avoid repeated events. Switch keeps log of the commands in local buffer.it filter/avoid repeated commands and followed exactly-once command property. Ravana provides transactional semantics by following event delivery,

processing and command execution and guaranteeing safety (execution of all events in errorless system and liveness (all command processed by switch and event will be processed by controllers).

3.3.6 Traffic Replication

Copying network traffic at the information plane level into all the controllers, is one approach of controller state replication. All switches will be joined with every one of the controllers all the while as per the approach followed in [67]. Fonseca request send to each one of the controllers, as controller gets a request from network device, it sends message arrival timestamp to all controllers. If each one of the controllers answer that none of them has a more settled timestamp, it send the response to the switch else the controller just updates its internal state.

Traffic replication rapidly get well from failures in a adversity scenario proposed by Gramoli et al. [70]. Two key matrices recovery time objective (RTO) and the recovery point objective (RPO) used by investigator. In case of disaster how much data lost, recovery time objective used for estimation of time of detection and mitigation of failure. While recovery point objective gives data of how many updates are lost.

Fast controller failover for multi domain SDN (FCF-M) proposed by Chan et al. [69] strong consistency achieved by each controller by replicating controller to its backup controller. Circular heartbeat mechanism used for detection of failure. Probe packet sending by each controller to its predecessor in heartbeat message mechanism, controller failover done locally if the backup of the controller available. Devices are distributed among controller based on its distance minimization and its capacity.

Centralized control [25] and distributed control [75] strategies considered in the design of control plane for providing fault tolerance. Control plane redundancy may require some sort of correspondence overhead, which affects scalability and performance. Distributed file system, distributed data structure and consensus protocol can be utilized to accomplish state replication.

3.4 Application/Control Interface

Network controller and applications share same execution space. So failure issues such as application failure isolation and control plane state fault tolerance discussed here.

3.4.1 Application Failure Separation

Almost existing available controllers [7, 20, 25] sharing same execution space, tightly coupled applications and network os with which leads to crash of controller on crash of applications. Additionally these controller don't display application resource consumption. So defective application may victimize shared resources. Fault tolerance and reliability must be taken care in the design of network operating system. As taken care in traditional operating system kernel design and architecture. In monolithic architecture, failure in one service leads to failure of the network controller. Mirco kernel architecture used independent service on the top of a kernel and connect internally using IPC communication interface.

3.4.2 Control Plane State Fault Tolerance

Logical centralized and physical SDN required harmonization of control planes with each other. Fault management done by backup replica in both centralized and distributed approach. In fault stage controller loses its state. Data plane and control plane failure both leads to failure of state of the controller, so both should be managed separately.

3.5 Application Layer

Testing and logical design of the application programs should be checked thoroughly before its implementation on application plane.

3.5.1 Application Design

Network requirement should be analyzed in the design phase first, Logical error in analysis never generate correct output. In the design phase fault management issues also considered, else bug, faults inserted in the since SDN application should be considered with specific knowledge domain, specific features should be used to provide fault tolerance constructs and abstractions. Few such deliberations are deadlocks, race condition etc. It requires high level fault management policies.

3.5.2 Application Correctness

For achieving desired output through application, comprehensive testing done on it. Correctness indicates desired output on valid input data. Application correctness can be checked by software verification and validation, former verification specialized tools used for providing domain specific knowledge.

3.6 Application/Control/Infrastructure Interface

Network accuracy infringements may generate errors at any plane or interface and network damage assessment consists all layers in order to successfully perform fault findings.

3.6.1 Network Precision Destruction

Network policies are translated into device configuration. Actual network behavior may violate security policies due to some experimental, faulty applications, routing, network controller mistranslation, Misinterpreted protocols. So network correctness monitored contently to notice possible destructions.

3.6.2 Network Investigating

When a disappointment happens, arrange heads must distinguish which applications are included and which succession of activities prompted the disappointment. This is certainly not a minor issue, considering that in vast systems an enormous number of reliant and free

events may happen amid a disappointment. Procedures to help arrange investigating incorporate chronicle and replaying system occasions [71], deciding least easygoing grouping [72] and breaking down system conduct by watching packet back-follows [73].

Distributed controllers layers are divided into application layers, application/control interface, control layers, control/infrastructure interface, data plane, app/control/infrastructure interface. Application layer described with the issue like application design, accuracy/correctness due to logical errors in API or in the adaptability. Distributed controllers contains combination of different application layers, modules. Link failures leads separate erroneous module from the whole system, that part will be down and not working but rest of the system should be fully functional. Control layers have collection of controllers. All are communicating with each other with east–west bound interfaces. Inter controller communication faces issues of load balancing, managing of distributed data store and on failure, replication technique need to be followed.

In Distributed controller, there may be chance of overload on one controller needs to migrate its switch to underloaded controller, aggregate load of the system will be overloaded and need to include more controller to balance the load and any controller may be shut down due to accidental error in hardware or software in all above cases fault generated. Fault may generate due to overload on controller. So Load balancing considered as fault management issue and switch migration techniques applied to solve it. Switch migration technique can be implemented as game theoretic approach [74], distributed decision [75], load informing strategy [76] and BalCon [77]—balanced controller approach. Details of all techniques given in next section. Replication technique are divided into active and passive replication [67, 78]. Flexibility and programmability are key capability of SDN. They meet the current network necessities such as multi-occupant cloud networks, versatile optical networks. SDN should solve fault management issues related with legacy network also.

4 Switch Migration in Distributed SDN Controller

In distributed controller domain, Switch migration is done in overloading of controller or flow requests of switch increases, heavily loaded switches should be migrated to lightly loaded controller. So dynamically change the relationship between switches of heavily

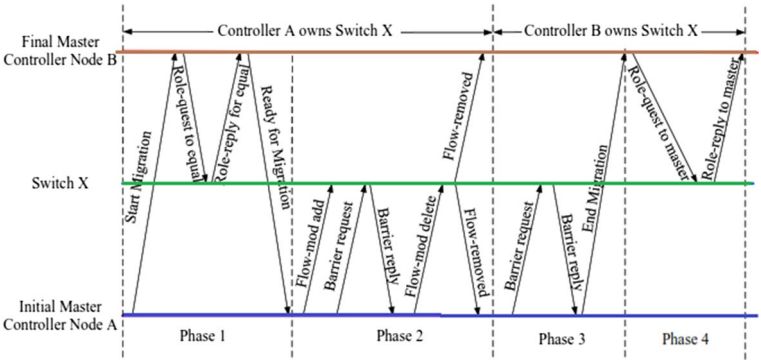

Fig. 3 Switch migration process [1, 2]

loaded controller to lightly loaded controller. Different methods used for switch migration (Fig. 3).

Figure 3 demonstrates a total depiction of the switch relocation system, which comprises of four stages. In stage 1, at first to wind up master controller B changes its part to equal. For it the underlying expert (A) sends a begin movement message to B through controller-to-controller channel. At that point, (B) sends Role-journeys to the change should have been relocated. After (B) gets Role-Reply from the switch, it informs (A) that the part changing has achieved. After (B) changes its part to level with, it gets nonconcurrent messages from the switch, however does not give a reaction. In stage 2, it embeds and expels a spurious flows. (A) Firstly sends Flow-mod to (X) to include another flow passage, which does not match with any packet. At that point, it sends another Flow-mod to erase the section. Consequently, the switch can send a Flow-evacuated message to controllers due to (B) is an equivalent controller at this moment. The Flow evacuated offers an exchange of proprietorship for the switch (X) from (A) to (B). Additionally, an obstruction message is asked for after the inclusion of the fake flow. In stage 3, it flushes pending solicitations for an obstruction. (A) Transmits a Barrier-ask for and sits tight for the Barrier-answer, simply after which it sends "end relocation" to the last master (B). In stage 4, it makes the objective controller last master. The last master (B) sets its part to master for the switch by sending a Role-ask for message to the switch. At long last, it refreshes the conveyed data store.

4.1 Elastic Control (ElastiCon)

ElastiCon [46] is the primary switch movement system in view of dynamic multicontroller design. Figure 4 sets the total system of ElastiCon, which contains three modules: load

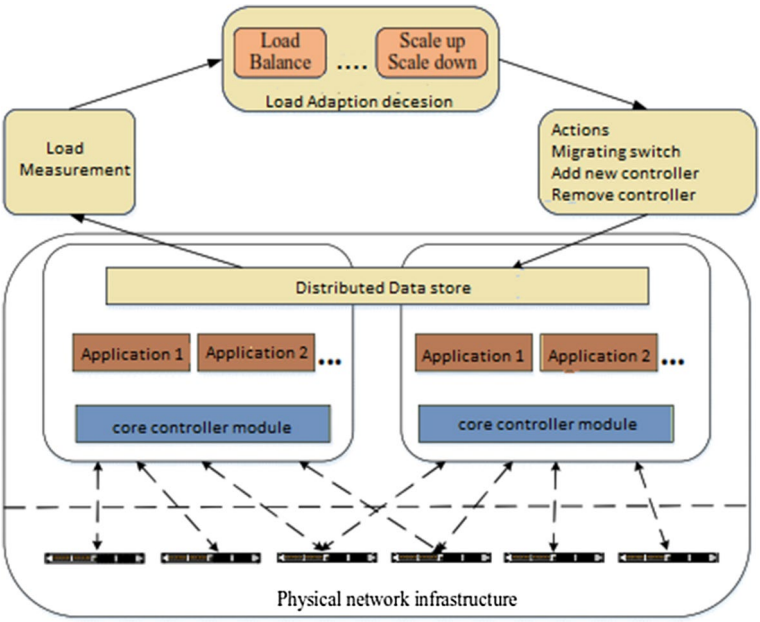

Fig. 4 Elastic framework [1]

estimation modules, load adjustment choice modules, and activity modules. The load estimation module gathers the load of every controller and sends the load data to load adjustment choice module, which chooses load distribution among controllers. The activity module conducts control activities (e.g., moving switch, including and evacuating controllers) to accomplish the dynamic control of controllers and switches. ElastiCon occasionally screens the load on every controller, distinguishes awkward nature, and naturally adjusts the load crosswise over controllers by moving a few changes from the over-burden controller to a softly stacked one. In the interim, with a specific end goal to orchestrate the movement, a novel switch relocation protocol is intended for empowering such load moving, which complies with the OpenFlow standard. At long last, a model of ElastiCon is manufactured and its execution is assessed in view of Mininet. In this way, ElastiCon guarantees unsurprising controller execution even under exceedingly unique workloads.

Yang, Zhou et al. [79] proposed answer for the adaptable system structure with decoupled control and information plane. Relocating switches can adjust the asset use of controllers and enhance network execution. Switch movement issue needs to date been defined as an asset usage expansion issue to address the scalability of the control plane. Be that as it may, this issue is NP-hard with high-computational complexities and without tending to the security difficulties of the control plane. They propose a switch relocation strategy, which translates switch movement as a mark coordinating issue and is figured as a 3-D earth mover's distance model to ensure deliberately essential controllers in the system. Thinking about the scalability, they additionally propose a heuristic technique which is time-proficient and reasonable to extensive scale systems. Simulation demonstrate that our proposed strategies can mask deliberately imperative controllers by decreasing the distinction of activity stack between controllers. In addition, our proposed techniques can essentially calm the movement weight of controllers and anticipate immersion assaults.

4.2 Game-Theoretic Approach

Chen et al. [74], the creators explain the switch movement calculation with diversion hypothesis. By taking light controllers as the amusement players and switches as the items, a zero-whole diversion demonstrate is abused to copy the rivalries for relocating switches among over-burden controllers. The controller chooses the ideal components to execute the exchange by expanding or diminishing the product estimation of the switch. The diversion display is quick and productive to accomplish switch movement yet is not reasonable for extensive scale organize because of the high unpredictability of calculation outline.

4.3 Distributed Decisions Scheme

Cheng et al. [75], the creators characterize the switch migration problem (SMP) and a network utility maximization (NUM) issue with the goal of boosting the quantity of serving demands under the accessible control asset. Conveyed hopping algorithm (DHA) is intended to accomplish ideal switch relocation through Log-Sum-Exp work. The DHA strategy is a period reversible markov chain process. The simulation result show DHA beats existing plans by decreasing stream setup time and enhancing the normal usage proportion of controller. They proposed scalable control component to choose which switch and where it ought to be relocated for an adjusted control plane. Each switch will be controlled by three distinct parts of controllers. Master, equal and slave. There is just a single master for the switch. The master can just get the switches' states yet additionally compose changes

to train the information plane. The equal controllers are acquainted with discrete burdens from the master. The slaves just read the conditions of the switches. Each switch could have in excess of one equivalent or slave controllers. On the off chance that master bombs because of over-burden or a few special cases, the equivalent controllers, or even slaves could be travelled to ace at the earliest opportunity. No component unequivocally demonstrating the switch movement or controller parts move, on the grounds that the authors of this specification feel this is the obligation of the controller to pick a master among themselves. The load of a SDN controller comprises of numerous components, for example, preparing of PACKET_IN events, keeping up the nearby domain view, speaking with different controllers, and additionally installing flow entries. In various situation, the extents of those components vary enormously. Be that as it may, the preparing of PACKET_IN occasions is for the most part viewed as the most conspicuous piece of the aggregate load. Likewise, the arriving rate of PACKET_IN occasions on a controller is checked to quantify its load.

4.4 Load Informing Strategy

Yu et al. [76], display a load adjusting system in view of a load advising technique for controllers. Decidedly, it constructs distributed choice architecture, including four parts that were load estimation, load illuminating, and adjusting choice and switch movement. In this procedure, every controller can occasionally effectively report its load data to different controllers, and it likewise handles and stores the load data from others. While the periodical dynamic load educating can diminish the choice deferral, it likewise causes extra preparing and correspondence overhead in the control plane. Particularly, when the present load esteem does not change much contrasted with the last esteem, revealing it to different controllers is an excess task.

4.5 Balanced Controller (BalCon)

BalCon is a heuristic solution proposed in [77]. One key limitation of distributed controller, with static mapping between switch and controller with uneven load distribution may rise among the controllers that may arise when network traffic condition may change during network operation. They proposed problem as mathematical optimization problem. It is NP-complete problem. Algorithm works runtime based on congestion analysis, when controller becomes overloaded, the algorithm finds a small strongly connected cluster of SDN switches to migrate such that overall control plane congestion is reduced. It depends on two key perception: (1) a compelling switch relocation ought to consider the correspondence examples of the SDN switches, (2) the switch movement ought to be prepared at the granularity of groups: switches with solid associations, which has the shorter remoteness to controller, ought to dependably be doled out to a similar controller. BalCon is accomplished by a reasonable model in view of Ryu, and the outcomes indicate BalCon altogether lessens the quantity of moving switches.

4.6 Load Fluctuating Based Synchronization (LVS)

Guo et al. [80] have proposed a controller state synchronization procedure named Load Variance based Synchronization (LVS), in order to redesign the execution of load

modifying in the multi-controller multi region SDN sort out. Interestingly with periodic synchronization based methods, LVS-based systems performed genuine state synchronizations among controllers while a load of a server outperforms a predefined constrain, which fundamentally restrains the synchronization overhead of controllers. The preliminary comes to fruition have exhibited that LVS gets authentic load modifying execution and circle free sending with less synchronization overhead, interestingly with existing systems. Nevertheless, two proposed LVS-based methodologies have not evaluated in a bona fide testbed.

4.7 SMDM (Switch Migration Based on Decision Making Scheme)

Wang et al. [81] proposed greedy calculation for switch relocation procedure. Proposed system will be utilized as a part of big business systems and WAN. They demonstrated that Dynamic switch movement is a promising way to deal with versatile scaling and load adjusting. By and by, switch movement happens in three cases. Right off the bat, if the accumulated traffic load goes past the limit all things considered, the new controllers ought to be included and the switches would be moved to them. Furthermore, as a controller is closed down or to rest for sparing correspondence cost and power, its switches ought to be relocated away. Thirdly, regardless of whether there is no adjustment in the quantity of sent controllers, switch movement activity must be performed by relocating chosen change to different controllers when an individual controller load is past its ability. It is known as load adjusting. They utilized switch movement trigger matric, the relocation productivity model to fabricate tradeoff between relocation expenses and load adjusting rate. Movement effectiveness show control the conceivable relocation activity. They execute evidence of the plan and present numerical assessment utilizing Mininet emulator to exhibit the adequacy of their proposition. SMDM plans appears (1) how to quantify load irregularity of controllers and choose whether to perform switch movement? (2) How to utilize movement arrange for that uses the relocation effectiveness model to manage the decision of conceivable relocation activities. They utilized total load an incentive to demonstrate genuine load data and give switch relocation procedure. Conventional SDN execution depends on centralized controller and have a few confinement related with execution and adaptability. Dispersed controllers approach can be utilized to take care of this issue. Authors et al. [82] give a system that modifies the quantity of dynamic controllers and delegates every controller. This system could limit flow setup time while acquiring low communication overhead. Be that as it may, it effectively prompts arrange reassignment since it needs to play out a reassignment of the whole control plane in light of the gathered activity measurements. They proposed logically centralized control plane, which could accomplish better scalability and reliability with independent controllers when an uneven enormous movement load touch base at these distributed controller.

4.8 Scalable and Crash Tolerant SDN Controller

Liang et al. [19] proposed group controller method for switch relocation calculation. They utilized unique load rebalancing strategy for grouped controllers. The numerous controllers utilized JGroup to organize the activity of switch movement. The total system partitioned into numerous groups and each group is setup one controller bunch. Anticipated strategy can progressively move the load over the various controllers through switch relocation. The component bolster controller failover without switch detachment maintaining a strategic

Table 4 An overview of current switch migration techniques for solving load balancing in distributed controller

Sr no.	Authors	Objective	Mode	Method	No of switches	Flows/second (Mbits/s)	Migration cost (ms)	Controller response time (ms)	Throughput (Mbit/s)	Simulation/evaluation
1	Wang et al. [81]	Improving migration efficiency by using switch migration trigger matric, migration efficiency model used migration costs and load balance rate. Used efficiency aware switch migration algorithm	Flat Beacon controller	MUMA (maximize resource utilization algorithm)	4	12.8	140	50	7	Switch migration scheme based on greedy algorithm to maximize the trade-off between migration costs and the load balance rate. an efficiency-aware migration algorithm based on greedy method was designed to utilize the migration efficiency model and thus guide the choice of possible migration actions
				DNMA(Distributed nearest migration algorithm)	4	12.8	105	65	4	
				SMDM(Switch migration based decision making)	4	12.8	100	47	2	
2	Liang et al. [83]	Scalable and crash tolerant SDN controller	Flat	Clustered controllers	8	15.32	20	0.9	15	Aggregate load of cluster should be collected before migration which increased the processing time

Table 4 (continued)

Sr no.	Authors	Objective	Mode	Method	No of switches	Flows/second (Mbits/s)	Migration cost (ms)	Controller response time (ms)	Throughput (Mbit/s)	Simulation/evaluation
3	Dixit et al. [1]	Elastic switch migration by adding/removing SDN controllers	Flat	Clustered controllers	8	55–135	0.35–2.8	0.21–0.76	155.4	Seamless migration of switches among multiple controllers How to select switches and target controllers not described
4	Yang Zhou et.al [79]	Elastic switch migration for control plane Load balancing in SDN	Flat	Optimal switch migration model and heuristic switch migration model	30	78.68	18	2.7	10	Simulation differentiate strategically important controllers by diminishing the difference of traffic load between controllers. Proposed method relives traffic pressure of controllers and prevent saturation attacks
5	Dixit et al. [46]	Achieving the dynamic mapping between switches and controllers	Flat	Linear programming	4	10	20	12	17	The controller response time has been reduced to 5 ms averagely

Table 4 (continued)

Sr no.	Authors	Objective	Mode	Method	No of switches	Flows/second (Mbits/s)	Migration cost (ms)	Controller response time (ms)	Throughput (Mbit/s)	Simulation/evaluation
6	Chen et al. [75]	Studying how to improve the load balancing performance of controllers in SDN	Flat	Game theory	100	10	22	30	15	Only 1.25% switches have been migrated when a half of controllers need master reelection operation
7	Cheng et al. [75]	Studying which switch should be migrated and where it will be moved	Flat	Heuristic approach	74	15	10	10	15	Only 10% of the switches in such as network have been migrated to load rebalance when there are 40~50% heavy controllers
8	Yu J et al. [76]	Controller take load balancing decision locally as rapidly, reducing time of load balancing	Flat	Linear programming	8	12.78	21	50	16	Result shows that load balancing is completed within 5 s The proposed method has higher throughput compared with static mapping between switch and controller

Table 4 (continued)

Sr no.	Authors	Objective	Mode	Method	No of switches	Flows/second (Mbits/s)	Migration cost (ms)	Controller response time (ms)	Throughput (Mbit/s)	Simulation/evaluation
9	Cello et al. [77]	Load balancing problem as mathematical optimization problem and based on congestion analysis	Flat	Heuristic approach	8	85	16.50	2.05	40	The proposed solution reduce the load imbalance among SDN controllers by 40% by migrating small number of switches. The computational time is 11.51 s in the proposed method
10	Hu, Tao et al. [84]	Load difference matrix and trigger factor used to measure load balancing on controllers, load balancing rate and migration cost considered for migration efficiency calculation	Flat	Efficiency aware switch migration approach	5–20	2	70	12	3.6	Reducing controller response time by 22%, improving controller throughput by 30% on average, maintaining good low migration costs and time, balancing rate

distance from the single purpose of disappointment issue. They executed a model framework in view of OpenDayLight Hydrogen controller to assess the execution of our plan. Primer outcome demonstrates that the technique empowers controllers to calm the overburden by means of switch movement and can enhance throughput and diminish the reaction time of the control plane. They intend to execute topology mindful switch movement calculation for enhancing scalability of the network.

Table 4 shows comparison of different switch migration techniques for load balancing. Different parameters such as flows/second, migration cost, controller response time, throughput are observed with respect to available different approaches taken by previous researchers. It is observed that Wang et al. [81] enhancing movement effectiveness by utilizing switch relocation trigger matric, migration efficiency model utilized movement expenses and load adjust rate. Utilized efficiency aware movement calculation. They took after MUMA (maximize resource utilization algorithm), DNMA (Distributed nearest movement algorithm) and SMDM (Switch migration based decision making). Switch relocation based in view of greedy algorithm to amplify the exchange off between movement costs and the load adjust rate. An effectiveness migration algorithm in light of greedy strategy was intended to use the migration proficiency model and hence manage the decision of conceivable relocation activities.

Liang et al. [83] utilizing scalable and crash tolerant group of controller. Total load of group of controller ought to be gathered before relocation which expanded the processing time. Yang Zhou et al. [79] followed optical and heuristic switch relocation model. Their Simulation differentiate deliberately important controllers by diminishing the difference of traffic load between controllers. Proposed method relives traffic pressure of controllers and prevent saturation attacks. Dixit et al. [1, 46] proposed flexible switch relocation for control plane load adjusting for achieving dynamic mapping between switches and controllers. Utilizing their strategy the controller reaction time has been decreased to 5 ms averagely, however they not portrayed how to choose switch and target controller.

Cheng et al. [74, 75] depicted which switch relocated and where it will be moved? They demonstrated that only 10% of the switches in such as network have been migrated to load rebalance when there are 40~50% overwhelming controllers. Yu et al. [76] depicted controller take load adjusting decision locally as quickly, reducing time of load balancing. Result shows that load balancing is completed within 5 s. The proposed method has higher throughput compared with static mapping between switch and controller. Cello et al. [77] proposed heuristic solution reduce the load imbalance among SDN controllers by 40% by relocating modest no of switches. Its computational time is 11.51 s. They take load balancing problem as mathematical optimization problem and based on congestion analysis. Hu, Tao et al. [84] described load difference matrix and trigger factor used to measure load balancing on controllers, load balancing rate and movement cost considered for migration efficiency calculation. They used efficiency aware switch migration approach. Simulation reveals reducing controller response time by 22%, improving controller throughput by 30% on average, maintaining good low migration costs and time, balancing rate.

5 Future Work—Fault Management in Distributed Controller

Analyzing surveyed effort we found future challenged in the fault management of distributed controller such as (1) providing same level of fault management as found in traditional legacy network(2) exploration of new possibilities in distributed SDN (3) integration of SDN with traditional network. Open issues discussed with different layers are as follows.

5.1 Data Plane

SDN permits novel one of a kind fault recovery arrangement, anyway recovery time in the coordination of various layers of SDN will be expanded. Indeed, even inheritance organize have issue of slow convergence, they are broadly actualized and completely tried through years, improving their heartiness and proficiency. In future dominant part of SDN testbed have more insight in information plane, incorporation between layers. In SDN numerous arrangements broaden the southbound protocol conduct as well as repurpose header fields (e.g. to utilize need field to allot reinforcement ways) so as to help fault recovery systems. In any case, the absence of standardization constrains the selection and common sense utilization of these arrangements. More up to date OpenFlow specifications give highlights identified with QoS and traffic checking, which can be utilized to help novel adaptation to non-critical failure components, despite the fact that these highlights are not straightforwardly identified with fault recovery.

Also, many switch sellers and system controllers just execute more established adaptations of OpenFlow protocol. In rundown, there is a gap between adaptation to internal failure endeavours and southbound protocol standardization, and between protocol specifications and accessible usage. Future specifications of OpenFlow and other southbound protocols, for example, OpFlex [4], may open new conceivable outcomes to fault tolerance research. Some extra capacities of information plane, for example, capacity to recognize navigated ways, while others propose new deliberations to help stateful information sending. This brings up some examination issues: How much, assuming any, insight ought to be set in the network devices? Which tradeoffs are included? In which cases is this suited? Initiatives like P4, a high level language for writing computer programs switches' packet processing, give more independency to information plane and enable more mind boggling rationale to be put in the network devices.

5.2 Control Plane

Most endeavors that utilization physical distribution use current methods, for example, appropriated file frameworks, to accomplish adaptation to non-critical failure. In any case, a gap that we identified in this approach is that it does not completely exploit SDN abilities. Methodologies more specific to SDN and systems administration may open new potential outcomes and accomplish preferred outcomes over more nonexclusive techniques. Most endeavors that utilization physical circulation use current strategies, for example, appropriated file frameworks, to accomplish adaptation to internal failure. In any case, a gap that we identified in this approach is that it does not completely exploit SDN abilities. Methodologies more specific to SDN and systems administration may open new potential outcomes and accomplish preferable outcomes over more nonspecific strategies.

Most distributed control plane designs share their state among copies. This approach, at most, enables applications to configure consistency level wanted. A more flexible

programming stage could offer control to organize applications define distinctive adaptation to internal failure strategies for various occasions and sorts of traffic. Also, the programming stage may bolster the definition of abnormal state adaptation to internal failure targets. For example, most recent variants of ONOS [6] enable software engineers to make goals, which speak to abnormal state control wants (e.g., availability between two has) that are interpreted and always authorized through low-level tenets.

5.3 Application Plane

Programming system arrangements, administrations, and indeed, even switch packet processing, brings numerous new potential outcomes to systems administration. In any case, we identified that couple of endeavors investigate.

What these conceivable outcomes convey to fault administration. One of the primary benefits of SDN is the likelihood of system administration through abnormal state terms. Numerous applications proposed for deliberations for example, network structure, modular composition of applications, and virtualization. Adaptation to internal failure reflections can be utilized to determine abnormal state adaptation to non-critical failure procedures and fault tolerant builds. Furthermore, many adaptation to non-critical failure components display some sort of trade off, e.g. network resilience vs. network performance. Deliberations can be given to permit specification of various strategies that would implement distinctive levels of adaptation to non-critical failure. A few endeavours as of now proposed techniques.

SDN can be incorporated in this way giving programmability and intelligent centralization for an incredible decent variety of situations, for example, 5G framework, Internet of-Things administration, virtual systems, remote systems. It is additionally conceivable to utilize different methodologies to use SDN abilities. For instance, Cui et al. [85] contend that enormous information preparing can be utilized to enhance organize execution by removing important data from a large number of system gadgets. Progressing research is as of now researching conceivable bearings [86, 87] to completely investigate SDN potential.

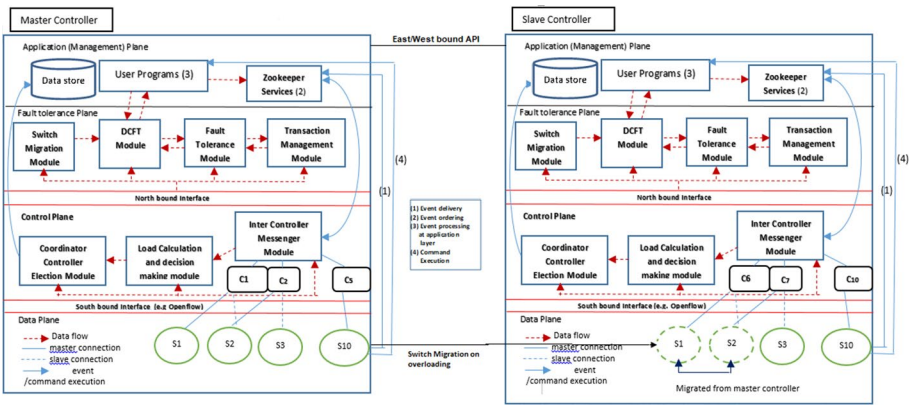

Fig. 5 Deployment of fault tolerance plane in SDN stack

6 Proposed Fault Tolerance Plane in the SDN Stack

Figure 5 shows the fault tolerance plane added in SDN stack. This plane is derived from the application plane. All the controllers of the clusters are calculating their own load at regular interval and send all the load details to the load calculation and decision making module. Coordinator controller will be decided through coordinator controller algorithm [88]. Two controllers are taken as a sample from the cluster of the controller. Master Controller can read/write the state of the switch. While slave controller can read only the state of the switch. 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 above deployment diagram 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.

All the modules shown in Fig. 5 will be implemented and tested with reference to scalability, consistency and reliability and finally derived a novel load balancing model which gives robust distributed SDN controller is the prime goal of our research.

Sequence of the event in the master and slave controllers are as (1) Switches generates events and forwarded to the controller. Such events are generated when switches receives packets or status of the port changes, (2) the controllers runs multiple applications that process and received events. Different events are ordered with zookeeper services which is used as East/westbound API also. (3) Events are processed by the multiple user programs at application layers (4) and replied to different switches as a command.

7 Conclusion

This work far reaching view on fault management in distributed controller. The paper identified fault management issues on each layers/interfaces. It is observed that that fault management issues raised by SDN are related to their layered engineering and logical centralization. In distributed controller Control plane have numerous controllers, overburden on any controller may prompt to failure of controller, node or link and it lead to fault. So we discussed solution of it as switch relocation of overloaded controller to underloaded one, different methods of switch migration compared with their flow/second, migration cost, controller response time, and throughput. Paper surveyed through classification of SDN controllers vide their adaptability, documentation, design choice (hierarchical/flat). Main concentration given on the faults generated because of overload on the controllers. Controllers are failed due to overload 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. A novel load balancing model including different module, will be proposed at the end of paper for generating robust distributed SDN controller.

References:s

1. Dixit, A. et al. (2013). Towards an elastic distributed SDN controller. In: ACM SIGCOMM Computer Communication Review. Vol. 43. No. 4. ACM.
2. Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., & Banerjee, S. (2011). DevoFlow: scaling flow management for high-performance networks. *ACM SIGCOMM Computing Communication Review., 41*, 254–265.
3. Yu, M., Rexford, J., Freedman, M. J., & Wang, J. (2010). Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computing Communication Review, 40*, 351.
4. OpenFlow switch specification 1.5.1, https://www.opennetworking.org/wpcontent/uploads/2014/10/openflow-switch-v1.5.1.pdf page no 74, accessed online on 13th Feb 2018.
5. Oktian, Y. E., et al. (2017). Distributed SDN controller system: A survey on design choice. *Computer Networks, 121*, 100–111.
6. Berde, P, et al. (2014) ONOS: towards an open, distributed SDN OS. In: Proceedings of the third workshop on hot topics in software defined networking. ACM, 2014.
7. Koponen, T., Martin, C., Natasha, G., Jeremy, S., Leon, P., Min, Z., et al. (2010). Onix: A distributed control platform for large-scale production networks. *OSDI, 10*, 1–6.
8. Tootoonchian, A., & Yashar, G. (2010) Hyperflow: A distributed control plane for OpenFlow. In: Proceedings of the 2010 internet network management conference on research on enterprise networking. 2010.
9. Medved, J., et al. (2014) Opendaylight: Towards a model-driven sdn controller architecture. In: 2014 IEEE 15th International Symposium on. IEEE, 2014.
10. Sakic, E., & Kellerer, W. (2017). Response time and availability study of RAFT consensus in distributed SDN control plane. *IEEE Transactions on Network and Service Management, 15*(1), 304–318.
11. Shalimov, A., et al. (2015) The Runos OpenFlow Controller. In:Software Defined Networks (EWSDN), 2015 Fourth European Workshop on. IEEE, 2015.
12. Sadasivarao, A., et al. (2013) Bursting data between data centers: Case for transport SDN. In: High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on. IEEE, 2013.
13. Big cloud fabric, [online] available: https://www.bigswitch.com/products/big-cloud-fabric. Accessed: July, 2018.
14. BORON, [online] available: https://www.opendaylight.org/what-we-do/current-release/boron. Accessed: July, 2018.
15. OSC, [online] available: https://www.sdxcentral.com/products/cisco-open-sdn-controller-osc/. Accessed: July, 2018.
16. Phemius, K., Mathieu, B., & Jeremie, L. (2014). Disco: Distributed multi-domain sdn controllers. In: Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014.
17. Hassas Yeganeh, S., & Yashar, G. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. In: Proceedings of the first workshop on hot topics in software defined networks. ACM, 2012.
18. Matsumoto, S., Hitz, S., & Perrig, A. (2014). Fleet: Defending SDNs from malicious administrators. In: Proceedings 3rd Workshop Hot Topics Software Defined Network, pp. 103–108.
19. Ferguson, A., Guha, A., Liang, C., Fonseca, R., & Krishnamurthi, S. (2013). Participatory networking: An API for application control of SDNs. In Proceedings ACM SIGCOMM Conference, pp. 327–338.
20. Katta, N., Zhang, H., Freedman, M., & Rexford, J. (2015). Ravana: Controller Fault-Tolerance in Software-Defined Networking. In: Proceedings of the ACM SIGCOMM Symposium on SDN Research, SOSR'15, (Santa Clara, CA, USA), June 2015.
21. Banikazemi, M., Olshefski, D., Shaikh, A., Tracey, J., & Wang, G. (2013). Meridian: An SDN platform for cloud network services. *IEEE Communications Magazine, 51*(2), 120–127.
22. Gude, N., et al. (2008). "NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review, 38*(3), 105–110.
23. J. McCauley et al. POX. [Online]. Available: https://github.com/noxrepo/pox. Accessed: 15 Jul 2018.
24. Ryu OpenFlow Controller. [Online]. Available: https://www.osrg.github.io/ryu. Accessed: 15 Jul 2018.
25. Erickson, D. (2013). The beacon OpenFlow controller. In: Proceedings 2nd ACM SIGCOMM Workshop Hot Topics Software Defined Networking 2013, pp. 13–18.
26. Salman, O., et al. (2016). SDN controllers: a comparative study. In: Electrotechnical Conference (MELECON), 2016 18th Mediterranean. IEEE, 2016.
27. Project Floodlight. Floodlight. [Online].Available: https://www.projectfloodlight.org/floodlight/. Accessed: July, 2018.

28. Lee, B., et al. (2014). IRIS: the Openflow-based recursive SDN controller. In: Advanced Communication Technology (ICACT), 2014 16th International Conference on. IEEE, 2014.
29. OpenMUL. High Performance SDN. [Online]. Available: https://www.openmul.org/. Accessed: July. 2018
30. Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., & Sherwood, R. (2012) On controller performance in software-defined networks. In: Proceedings 2nd USENIX Conference Hot Topics Management Internet Cloud Enterprise Network. Services, 2012.
31. Zhang, Y., et al. (2017). A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications, 103*, 101–108.
32. VMware NSX, [online] available:https://www.sdxcentral.com/vmware/definitions/what-is-vmware-nsx/. Accessed: July, 2018.
33. Programmable flow controller NEC, [online] available:https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/nec-programmableflow-controller/. Accessed: July, 2018.
34. vneio/sdnc, [online] available: https://github.com/vneio/sdnc/wiki. Accessed: July, 2018.
35. Cherry, [online] https://github.com/superkkt/cherry. Accessed: July 2018
36. Opencontrail, [online] available: https://www.opencontrail.org/opencontrail-quick-start-guide/. Accessed: July, 2018.
37. Faucet, [online] available: https://github.com/faucetsdn/faucet. Accessed: July, 2018.
38. VSC, [online] available: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/nuage-networks-vsc/. Accessed: July, 2018
39. VortiQa, [online] available: https://www.sdxcentral.com/products/freescale-vortiqa-open-network-on-director-software/. Accessed: July, 2018.
40. B4N, [online] available: https://brain4net.com/products/. Accessed: July, 2018.
41. Fu, Y., et al. (2014) Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks. In: 2014 IEEE 22nd International Conference on Network Protocols (ICNP). IEEE, 2014.
42. Jain, S., et al. (2013). B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computing Communication Review, 43*(4), 3–14.
43. Lei, T., et al. (2014). SWAN: An SDN based campus WLAN framework. In: Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronics Systems (VITAE), 2014 4th International Conference on. IEEE, 2014.
44. Arbettu, R.K., et al. (2016). Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers. In: Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International. IEEE, 2016.
45. Chang, Y., et al. (2017). Hydra: leveraging functional slicing for efficient distributed SDN controllers. In: Communication Systems and Networks (COMSNETS), 2017 9th International Conference on IEEE, 2017.
46. Dixit, A., Hao, F., Mukherjee, S., Lakshman, T.V., & Kompella, R.R. (2014). ElastiCon; an elastic distributed SDN controller. In: 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Marina del Rey, CA, 2014, pp. 17–27.
47. Botelho, F., et al. (2014). Smartlight: A practical fault-tolerant SDN controller. arXiv preprint arXiv:1407.6062.
48. van Adrichem, N. L., Van Asten, B. J., & Kuipers, F. A. (2014). Fast recovery in software-defined networks. In Software Defined Networks (EWSDN), 2014 Third European Workshop on, (Budapest, Hungary), pp. 61–66, IEEE, Sept. 2014.
49. Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2011) Enabling fast failure recovery in OpenFlow networks. In: Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the, pp. 164–171, IEEE, 2011.
50. Vinoski, S. (2006). Advanced message queuing protocol. *IEEE Internet Computing, 6*, 87–89.
51. Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. In: Proceedings of the IEEE, vol. 103, pp. 14–76, Jan 2015.
52. Kreutz, D., Ramos, F.M., & Verissimo, P. (2013). Towards secure and dependable software-defined networks. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13, (New York, NY, USA), pp. 55–60, ACM, Aug. 2013.
53. da Silva, A. S., Smith, P., Mauthe, A., & Schaeffer-Filho, A. (2015). Resilience support in software-defined networking: A survey. *Computer Networks, 92*(1), 189–207.
54. Chen, J., Chen, J., Xu, F., Yin, M., & Zhang, W. (2015). When Software Defined Networks Meet Fault Tolerance: A Survey. In: Algorithms and Architectures for Parallel Processing: 15th

International Conference, ICA3PP 2015, (Zhangjiajie, China), pp. 351–368, Springer International Publishing, Nov. 2015.
55. Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2013). OpenFlow: Meeting carrier-grade recovery requirements. *Computer Communications, 36*(6), 656–665.
56. Fernandes, E., & Rothenberg, C. (2014) OpenFlow 1.3 software switch. In: Brazilian Symposium on Computer Networks and Distributed Systems, 2014.
57. Doria, A., et al. (2010) Forwarding and control element separation (ForCES) protocol specification. In: Internet Engineering Task Force, 2010.
58. Levin, D., Wundsam, A., Heller, B., Handigol, N., & Feldmann, A. (2012). Logically Centralized?: State Distribution Trade-offs in Software Defined Networks. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networking, HotSDN'12, (New York, NY, USA), pp. 1–6, ACM, Aug. 2012.
59. Hunt, P., Konar, M., Junqueira, F.P., & Reed, B. (2010) Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX'10 Annual Technical Conference.
60. Heller, B., Sherwood, R., & McKeown, N. (2012). The controller placement problem. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networking, HotSDN'12, (New York, NY, USA), pp. 7–12, ACM, Aug. 2012.
61. Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review, 44*(2), 35–40.
62. Johns, M. (2015). *Getting Started with Hazelcast*. Birmingham: Packt Publishing Ltd.
63. Alliance, O. (2003). *OSGI service platform, release 3*. Clifton: IOS Press Inc.
64. Spalla, E.S., Mafioletti, D.R., Liberato, A.B., Rothenberg, C., Camargos, L., VillaÃga, R.D.S., & Martinello, M. (2015). Resilient Strategies to SDN: An Approach Focused on Actively Replicated Controllers. In: Computer Networks and Distributed Systems (SBRC), 2015 XXXIII Brazilian Symposium on, pp. 246–259, May 2015.
65. OpenReplica Coordination Service. https://openreplica.org/. Accessed: 30 Jun 2018.
66. Lamport, L. (2001). Paxos Made Simple. *ACM SIGACT News, 32*(4), 34.
67. Fonseca, P., Bennesby, R., Mota, E., & Passito, A. (2012) A replication component for resilient openflow-based networking. In: Network Operations and Management Symposium, NOMS'12, (Maui, HI, USA), pp. 933–939, IEEE, 2012.
68. Pashkov, V., Shalimov, A., & Smeliansky, R. (2014) Controller failover for SDN enterprise networks. In: International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), pp. 1–6, Oct 2014.
69. Chan, Y.-C., Wang, K., & Hsu, Y.-H. (2015) Fast controller failover for multidomain software-defined networks. In: Networks and Communications (EuCNC), 2015 European Conference on IEEE, 2015, pp. 370–374.
70. Gramoli, V., Jourjon G., & Mehani, O. (2015) Disaster-tolerant storage with SDN. In: International Conference on Networked Systems, pp. 293–307, Springer, 2015.
71. Wundsam, A., Levin, D., Seetharaman, S., Feldmann, A., et al. (2011) OFRewind: Enabling Record and Replay Troubleshooting for Networks. In: USENIX Annual Technical Conference, (Portland, OR, USA), June 2011.
72. Scott, C., Wundsam, A., Whitlock, S., Or, A., Huang, E., Zarifis, K., & Shenker, S. (2013) How did we get into this mess? Isolating fault-inducing inputs to SDN control software. Tech. Rep. UCB/EECS-2013–8, EECS Dept., University of California, Berkeley, 2013.
73. Handigol, N., Heller, B., Jeyakumar, V., Maziéres, D., & McKeown, N. (2012). Where is the debugger for my software-defined network? In: Proceedings of the first workshop on Hot topics in software defined networking, pp. 55–60, ACM, Aug. 2012.
74. Chen, H., Cheng, G., & Wang, Z. (2016). A game theoretic approach to elastic control in software-defined networking. *China Communication, 13*(5), 103109.
75. Cheng, G, Chen, H, & Wang, Z, et al. (2015) DHA: distributed decisions on the switch migration toward a scalable SDN control plane. In: IFIP Networking Conference. IFIP, 2015, pp. 473–477.
76. Yu, J., Wang, Y., & Pei, K., et al. (2016). A load balancing mechanism for multiple SDN controllers based on load informing strategy. In: Network Operations and Management Symposium, 2016, pp. 1–6.
77. Cello, M., Xu, Y., & Walid, A., et al. (2017). BalCon: A distributed elastic SDN control via efficient switch migration. In: IEEE International Conference on Cloud Engineering, 2017, pp. 40–50.
78. Bessani, A., Sousa, J., & Alchieri, E. E. P. (2014). State machine replication for the masses with bft-smart. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 355–362, June 2014.

79. Zhou, Y., et al. (2018). Elastic switch migration for control plane load balancing in SDN. *IEEE Access, 6*, 3909–3919.
80. Guo, Z., et al. (2014). Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks, 68*, 95–109.
81. Wang, C., Hu, B., Chen, S., Li, D., & Liu, B. (2017). A switch migration-based decision-making scheme for balancing load in SDN. *IEEE Access, 5*, 4537–4544.
82. Fonseca, P., & Edjard, M. (2017). A survey on fault management in software-defined networks. *IEEE Communications Surveys and Tutorials, 19*, 2284–2321.
83. Liang, C., Kawashima, R., & Matsuo, H. (2014). Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers. In: Proceedings 2nd Int. Symp. Comput. Netw. (CANDAR), 2014, pp. 171–177.
84. Hu, T., et al. (2017). EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking. *Peer-to-Peer Networking and Applications, 12*, 1–13.
85. Cui, L., Yu, F. R., & Yan, Q. (2016). When big data meets software-defined networking: SDN for big data and big data for SDN. *IEEE Network, 30*, 58–65.
86. Hakiri, A., & Berthou, P. (2015). Leveraging SDN for the 5G networks: Trends, prospects and challenges. CoRR, vol. abs/1506.02876.
87. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., et al. (2013). B4: Experience with a globally-deployed software defined WAN. *SIGCOMM Computer Communication Review, 43*, 3–14.
88. Lakhani, G., & Kothari, A. (2020). Coordinator controller election algorithm to provide failsafe through load balancing in Distributed SDN control plane. In: Proceedings of the 1st Springer CCIS series conference, COMS2, March 2020.

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

Dr. Amit D. Kothari has completed his PhD from Hemchandracharya North Gujarat University, Patan,Gujarat in 2011. During Ph.D. his research area was routing protocols for Mobile Adhoc Network (MANet). Six students are pursuing their Ph.D. research work under him. More than 13 National/International research paper are published Delivered expert talk at many universities during 15 years of professional carrier. His interest area is—Computer Network, Network/Information Security, Parallel/Distributed Computing and artificial intelligence. He offers his service as a book reviewer for McGrew Hill publication. He is invited as member of Doctorate review Committee at many universities.