Journal of Network and Computer Applications 103 (2018) 101–118

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Review

A survey on software defined networking with multiple controllers

Yuan Zhang[a], Lin Cui[a,b,*], Wei Wang[c], Yuxiang Zhang[a]

[a] Department of Computer Science, Jinan University, Guangzhou, PR China
[b] Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, PR China
[c] Network and Information Center, Shandong University at Weihai, Weihai, PR China

ARTICLE INFO

Keywords:
Software defined networking
Multiple controllers
OpenFlow

ABSTRACT

Compared with traditional network, Software Defined Networking (SDN) decouples control plane and data plane, providing programmability to configure the network. In spite of such capability, one of the criticisms of SDN is that the SDN controller is a single point of failure and hence the controller decreases overall network availability. Having multiple controllers improves reliability of the network because the data plane can continue to operate if one controller fails. Furthermore, a single controller of SDN has many limitations on both performance and scalability. Thus, multiple controllers are required and critical for large-scale networks. However, multiple controllers increase network complexity dramatically and impose many new challenges to the management and schedule of SDN. This paper surveys latest researches on multiple controllers of SDN. Benefits and challenges of multiple controllers are discussed after giving an overview of SDN and OpenFlow in the paper. Afterward, we dwell on the detailed design principles and architectures of SDN with multiple controllers. Following that, current research works on multiple controllers placement and scheduling are carefully summarized and analyzed. Finally, we conclude this survey paper with some future works and suggested open research directions.

1. Introduction

Traditional Internet architecture is sufficiently complex and closed, making it difficult for network administrators to operate and manage when considering network dynamic and various application requirements. Hence, Software Defined Networking (SDN) (Farhady et al., 2015; Nunes et al., 2014; Masoudi and Ghaffari, 2016; Li et al., 2016) has been proposed as a promising network paradigm to solve those problems and accelerate innovation. SDN decouples the control plane and data plane to achieve a logically centralized control architecture, providing programmability to configure the network (Lopes et al., 2016). SDN controller can obtain global information of the whole network, which is convenient for operators and researchers to adjust the network flexibly and extend new network functions.

Different from traditional network, by separating the control plane and data plane in SDN, control planes are merged into a single entity named controller. As a decision maker, SDN controller is able to provide Application Program Interfaces (APIs) to upper applications and allow operators to deploy various network policies flexibly, according to scenarios and applications requirements. Therefore, controllers play an important role in SDN.

The control plane of SDN may take different forms, either centralized (e.g., single controller) or distributed (e.g., multiple controllers) architecture. Though previous study (Heller et al., 2012) has shown that a single controller may be sufficient for many medium-size networks, multiple controllers are still required in many scenarios (Jain et al., 2013; Berman et al., 2014; Al-Fares et al., 2010) considering the efficiency, scalability and availability of the networks. With multiple controllers, response latency of requests can be reduced through load balancing and avoiding overloaded controllers (Karakus and Durresi, 2017). New controllers can be added dynamically to achieve higher performance in order to meet the needs of the networks. Moreover, multiple controllers also provide redundancy mechanism to avoid single point of failure and improve the security of the control plane (Shalimov et al., 2013; Yan et al., 2016; Oktian et al., 2017).

Multiple controllers have been discussed in previous works. For instance, Nunes et al. (2014) survey the state-of-the-art in programmable network with an emphasis on SDN and compare centralized architecture with distributed scheme. Oktian et al. (2017) focus on the design choice of distributed SDN controller, showing that building distributed controller should consider the following aspects: scalability, robustness, consistency and security. Kreutz et al. (2014) present a comprehensive survey on SDN. They compare and summarize the platforms of centralized controllers and distributed controllers, and analyze different

* Corresponding author at: Department of Computer Science, Jinan University, Guangzhou, PR China.
E-mail addresses: michellinyuan@gmail.com (Y. Zhang), tcuilin@jnu.edu.cn (L. Cui), wwei@sdu.edu.cn (W. Wang), samuelzyx0924@gmail.com (Y. Zhang).

https://doi.org/10.1016/j.jnca.2017.11.015
Received 25 May 2017; Received in revised form 3 November 2017; Accepted 28 November 2017
Available online 08 December 2017
1084-8045/

challenges of distributed controller platforms. Xia et al. (2015) discuss multiple controllers with a focus on controller design. Chauhan et al. (2016) dwell on the architecture of SDN with multiple controllers, and discuss the difference between logically centralized architecture and logically distributed architecture. Wibowo et al. (2017) present a review on research status of SDN, discussing the challenges in multi-domain SDN controller design and architectures.

In this paper, a comprehensive survey on multiple controllers of SDN is presented, ranging from fundamental techniques to the scheduling issues on multiple controllers. The main contributions of this paper are as follows:

- A review of multiple controllers challenges and benefits, and a table with comparison of existing multiple controllers platforms and projects.
- Analysis of design principles for multiple controllers from three aspects: consistency, fault tolerance and availability.
- A review of current research into multiple controllers architecture, placement and scheduling, and the major research issues that can be considered in future.

The taxonomy of multiple controllers discussed in this paper is shown in Fig. 1. Four aspects of multiple controllers are discussed, including multiple controllers design principles, architecture, placement and scheduling. All of these issues are essential and correlated factors that together determine the performance of multiple controllers. Design principles need to be implemented in multiple controllers architecture to realize high performance platforms. Especially, efficient consistency solutions can improve the controllers-to-controllers communication. For multiple controllers placement, several metrics should be taken into account. Some metrics are related to design principles, e.g., fault tolerance and reliability. Due to dynamic traffic in networks, scheduling strategies should be implemented for load balance and efficiency. By allowing dynamic controllers assignment, controllers respond time and communication load between controllers and switches can be reduced.

The remainder of the paper is organized as follows: Section 2 briefly provides an overview of SDN architecture and OpenFlow. Section 3 discusses the benefits and challenges of multiple controllers for SDN. Section 4 lists some existing multiple controller platforms and summarizes their features. Multiple controllers design principles are explained in Section 5. Further, multiple controllers architectures are discussed in Section 6. Section 7 provides an in-depth analysis of the placement of multiple controllers considering multiple metrics. Scheduling for multiple controllers are presented in Section 8. Finally, Section 9 summarizes several issues for future research on multiple controllers and Section 10 briefly concludes this paper.

2. Overview of SDN and OpenFlow

In this section, we briefly outline the concepts and the basic theories of SDN, including SDN architecture, SDN controllers and a typical southbound interface, i.e., OpenFlow (McKeown et al., 2008).

2.1. SDN architecture

SDN is an emerging networking paradigm to improve the performance of current network architecture (Kim and Feamster, 2013). SDN breaks the vertical integration by separating control plane from data plane. With such separation, switches become simple forwarding devices and the control logic can be implemented in a logically centralized controller. The controller is responsible for policy enforcement, network configuration, topology management, link discovery, flow table entry and so on. SDN architecture normally contains three

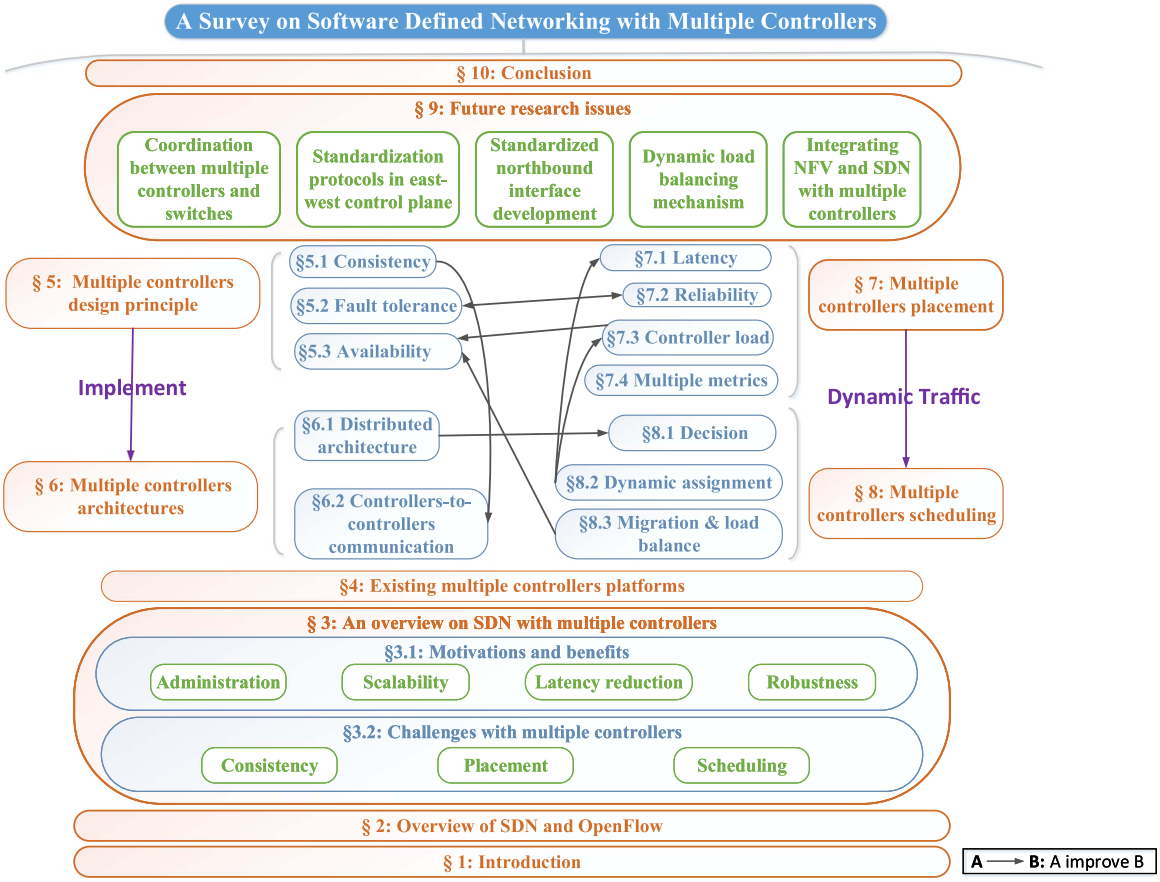

Fig. 1. Condensed overview of this survey on SDN with multiple controllers.

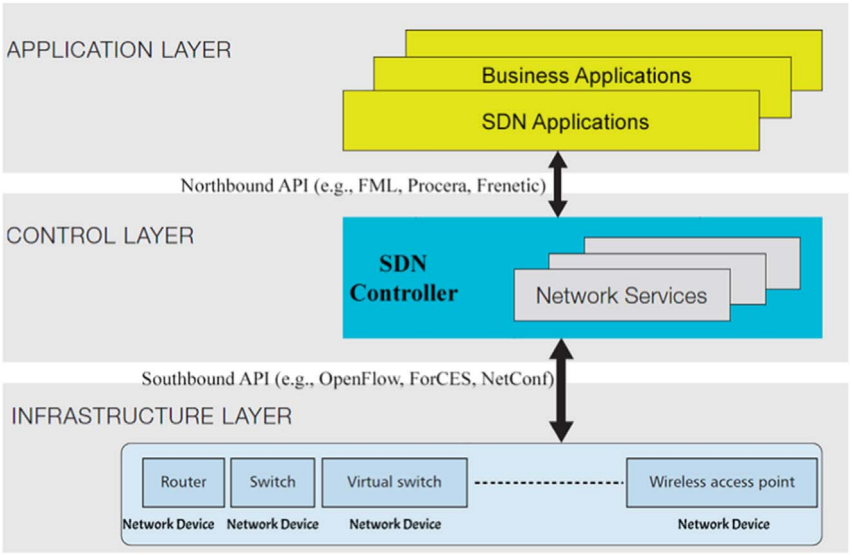

Fig. 2. A simplified view of SDN architecture (SDN architecture Overview, 2014).

layers (i.e., infrastructure layer, control layer and application layer) and two interfaces (i.e., southbound interface and northbound interface). Fig. 2 shows a simplified SDN architecture (SDN architecture Overview, 2014):

- **Infrastructure layer** is composed of SDN-enabled switches and other network elements (NE). Each network element in logic represents all or part of physical network resources. The infrastructure layer provides flow forwarding information to the upper controllers through the data-control plane interface, also known as southbound APIs. The infrastructure layer forwards traffic according to decisions made by the control layer. The switches consist of three parts, namely secure channel, flow table and OpenFlow protocol (McKeown et al., 2008). Secure channel is the interface that connects each OpenFlow switch to controller. Flow table is an assemblage of forwarding policies. The OpenFlow protocol is a standard southbound API between the control plane and the data plane of SDN architecture.
- **Southbound APIs** are communication interfaces that connect the infrastructure layer and the control layer. Its main function is to enable communication between the SDN controllers and the network elements, in order to discover the network topology, push the flow table and implement the control policies. The OpenFlow protocol is seen as the most popular southbound API for SDN.
- **Control layer** is the core component of SDN, which is responsible for network management, consisting of one or multiple controllers. Controllers usually have global networks information to determine appropriate network policies and then configure switches through a secure channel. Controllers have three interfaces, namely: (1) northbound APIs, for the communication between applications and controllers; (2) southbound APIs, for the communication between controllers and switches; and (3) east-west interface, for the communication among multiple controllers.
- **Northbound APIs** are communication interfaces that link the control layer and the application layer. The goal is to enable applications to easily manage network resources and capabilities. Different from southbound APIs, northbound APIs still lack standard specification, due to that existing controllers and applications have diverse features, which is difficult to unify (Hakiri et al., 2014).
- **Application layer** contains SDN applications and services that are designed to meet user requirements. And it also can access and control switches in data plane through the control layer and communicate with the control layer through the northbound APIs.

2.2. OpenFlow protocol

OpenFlow is a widely adopted protocol that defines an open standard southbound API for SDN. Most current controllers implementation are compliant with the OpenFlow protocol, e.g., POX (Kaur et al., 2014), NOX (Gude et al., 2008), Beacon (Jiang et al., 2014), and Floodlight (Morales et al., 2015). And many network switch and router vendors have released switches for OpenFlow, including Alcatel-Lucent, Big Switch Networks, Brocade Communications, Arista Networks, Pica8, Huawei, Cisco, IBM, Juniper Networks and so on.

OpenFlow standards are managed by the Open Networking Foundation (ONF) (Open Networking Foundation, 2011), which is an organization dedicated to promotion and adoption of SDN/NFV. So far, ONF has published OpenFlow l.5.0 at the end of 2014. OpenFlow 1.0.0 (OpenFlow Specification 1.0, 2009) version defines flow table with 12 tuples, e.g., source/destination IP address, source/destination MAC address. OpenFlow 1.1.0 (OpenFlow Specification 1.1.0, 2011) increases some rules and supports multiple flow tables, group tables, action sets and other functions. Starting from OpenFlow 1.2.0 (OpenFlow Specification 1.2.0, 2011), OpenFlow supports IPv6 source/destination address and multiple controllers. Network congestion has been one of stubborn problems that Internet is faced with. Therefore, OpenFlow supports flow congestion control mechanism starting from version 1.3.0 (OpenFlow Specification 1.3.0, 2012). In version 1.4.0 (OpenFlow Specification 1.4.0, 2013), OpenFlow increases flow table deletion and replication mechanism. ONF proposed OpenFlow-Configuration (OF-CONFIG) protocol (OpenFlow-Configuration Protocol, 2014) as a configuration protocol. OpenFlow 1.5.0 (OpenFlow Specification 1.5.0, 2014) adds the new “Egress table”, and allow matching packet based on its output port. Table 1 shows main the information of different versions of OpenFlow protocol.

As shown in Table 1, starting from version 1.2.0, OpenFlow protocol defines a multi-controller scheme to improve reliability of communications between the controllers. In version 1.2.0, OpenFlow proposed that a switch can establish communication with multiple controllers to improve reliability of the control plane.

From OpenFlow 1.0.0 to 1.5.0 version, OpenFlow protocol is improving constantly. Development and evolution of OpenFlow protocol aim at two aspects. One is to enhance control layer, making the system much more richer and flexible. Another is to improve the capability of the infrastructure layer to match more keywords and perform more actions. Open Virtual Switch (OVS) and of13softswitch are common implementations of SDN switches. OVS supports

Table 1
Development of OpenFlow protocol.

OpenFlow version	Date	Features
OpenFlow 1.0.0 OpenFlow Specification 1.0 (2009)	Dec,2009	Fundamental architecture, single flow table, IPv4
OpenFlow 1.1.0 OpenFlow Specification 1.1.0 (2011)	Feb,2011	Multiple flow table, group table, MPLS and VLAN
OpenFlow 1.2.0 OpenFlow Specification 1.2.0 (2011)	Dec,2011	IPv6, multiple controllers
OpenFlow 1.3.0 OpenFlow Specification 1.3.0 (2012)	Jun,2012	Single flow measure, IPv6 extend header
OpenFlow 1.4.0 OpenFlow Specification 1.4.0 (2013)	Oct,2013	Flow table synchronization mechanism, bundling message
OpenFlow 1.5.0 OpenFlow Specification 1.5.0 (2014)	Dec,2014	Data packet type identification process, egress table, scheduled bundle expansion

OpenFlow 1.5.0 and Of13softswitch (of13softswitch, 2013) supports OpenFlow 1.3.0. So both of them can support multiple controllers. Multiple controllers can be connected to an OVS or of13softswitch after configuring related information.

OpenFlow protocol allows switches to keep connection with multiple controllers at the same time (Spalla et al., 2015). For each switch, the controllers can be in three states, *Equal*, *Slave* and *Master*. *Equal* is the default state for controllers, in which they have all control permissions to switches. A switch can send all types of asynchronous messages (such as Packet-In and Flow-Remove) to the controllers that are in the *Equal* state. The controller in *Equal* state can also send control commands to the switch to change switch state, and send a state change request to the switch to change its state into *Slave* or *Master*. The controller in *Slave* state can take read-only operations on switches. *Master* controllers have the same control authority as *Equal*. The difference is that there is only one *Master* controller for each switch. So, when a controller becomes the *Master* of a switch, the switch will change the previous *Master* controller into *Slave*. Each switch can connect to multiple *Equal* controllers and *Slave* controllers at the same time, but only one *Master* controller is allowed to be connected.

Although OpenFlow is the most popular southbound interface in SDN, it still can not provide flexible programmability. P4 (Bosshart et al., 2014) is proposed as a new network programming language, which makes up the gap.

3. Multiple controllers: benefits and challenges

In SDN, a single control plane brings many benefits such as controlling the network by a central node and abstracting the underlying network infrastructure easily. However, a single controller imposes potential issues of scalability and reliability. Hence, the deployment of multiple controllers is a critical requirement (Li et al., 2017). In this paper, SDN with multiple controllers refers to the SDN-enabled network that has more than one controller working together for high performance, scalability, availability, etc. Multiple controllers introduce both benefits and challenges to SDN networks.

3.1. Motivations and benefits

Large-scale and heterogeneous SDN networks may need to be divided into several domains (Schmid and Suomela, 2013), and each domain needs at least one control entity to supervise it. In such scenario, multiple controllers are required to manage the whole network. Having multiple controllers in the network can improve stability, as the switches can continue operating if one controller or controller connection fails. Multiple controllers can be implemented in distributed control plane, which means logically centralized but physically distributed. There are other benefits to use multiple controllers, especially distributed control plane (Schmid and Suomela, 2013).

- **Administration.** Efficient administration is an important motivation for multiple controllers. The capability of a single controller is limited compare to multiple controllers. It is difficult for a single controller to manage a large-scale network with multiple domains (Hong et al., 2013). Due to the different features in different controllers, the collaboration of multiple controllers can improve management efficiency of multi-domain network or heterogeneous network (Fratczak et al., 2013).
- **Scalability.** Current OpenFlow approach may lead to heavy load on a single controller, which reduce scalability of the network. Multiple controllers architecture can be a practical solution so that controllers can be added or removed dynamically. Besides, the requests can be load balanced among multiple controllers to avoid bottleneck at a single controller (Yeganeh et al., 2013). And, multiple controllers can be an appropriate architecture in supporting a large number of tenants (Bozakov and Papadimitriou, 2012).
- **Latency reduction.** There are many latency-sensitive applications upon the controllers. If every event is responded by a single controller in successively sequence, it may cause the controller overload and increase response latency. With multiple controllers, the requests can be load balanced among different controllers or forwarded to the most appropriated one to reduce burst traffic processing latency.
- **Robustness**. SDN architecture with multiple controllers adopts multi-points backup, to avoid single point of failure, and enhance the network robustness. When a controller fails, other controllers can pick up the load with little or no changes so as to maintain network performance and stability. Multiple controllers can reduce the probability of network failures, especially controller faults.

3.2. Challenges with multiple controllers

Although SDN with multiple controllers has been introduced as a troubleshooter to a single controller, it also brings some new challenges that need to be resolved (Lange et al., 2015). For instance, inconsistency among multiple controllers may increase response latency or cause errors. Challenges imposed by multiple controllers are summarized as follows:

- **Consistency.** When implementing multiple controllers platform, *how to synchronize the network state information among controllers* is a critical problem, also known as the controllers consensus problem (Zhang et al., 2017; Liu et al., 2015). Many distributed control plane platforms of SDN with multiple controllers rely on consensus algorithms such as Paxos (used by ONIX (Koponen et al., 2010)) and Raft (used by ONOS (Berde et al., 2014)). However, current consensus mechanisms are not perfectly suitable due to implementation complexity, low availability and incremental latency (Panda et al., 2017). Consistency is also regarded as a essential design principle for multiple controllers (see more details in Section 5.1).
- **Placement.** When deploying multiple controllers in networks, two essential questions need to be taken into consideration (Heller et al., 2012): *how many controllers are needed?* And *where in the topology should they go?* These two issues are all NP-hard

problems, but they play a significant role in implementing SDN multiple controllers.[1] Other aspects of placement problem (e.g., latency and load) are explained in Section 7.

- **Scheduling.** SDN networks performance can be improved with the help of multiple controllers (Xu et al., 2017), but some inevitable issues still exist: *how to schedule multiple controllers to prevent one of the controllers overload?* And *how to make an overload controller balanced as quickly as possible?* The control plane should make an adaptation to uneven load distribution rapidly, when mapping between a switch and a controller is statically configured. More details are discussed in Section 8.

3.3. Summary

In this section, multiple controllers benefits and challenges are discussed. Multiple controllers provide efficient administration, scalability, latency reduction and robustness. However, several challenges, such as consistency among controllers, controllers placement problem and multiple controllers scheduling need to be resolved.

4. Existing multiple controllers platforms

Various multiple controllers platforms and projects have been proposed. Different multiple controllers platforms have their own features, which are suitable for different applications. Table 2 compares the representatives of the existing multiple controllers platforms and projects, from five aspects: development languages, baseline controller, organization and whether it is open source and their own features. Especially, the web links of the open source projects are also provided.

- HyperFlow (Tootoonchian and Ganjali, 2010) is an event-based distributed controller architecture. It is the first distributed controller architecture that supports OpenFlow.
- Onix (Koponen et al., 2010) performs better in reliability, scalability and universality. Onix consists of physical network infrastructure, network connection infrastructure and network control logic, using Network Information Base (NIB) to maintain network state. Three kinds of strategies including partition, aggregation and consistency & stability are used to realize the scalability of control plane.
- Kandoo (Hassas Yeganeh and Ganjali, 2012) is a hierarchical distributed SDN control plane. The basic idea of Kandoo is that local controller handle frequent networking events from local applications without global information, while root controller handles non-local applications. Kandoo processes local application flows and global application flows separately, which avoids potential bottleneck in terms of scalability. However, Kandoo restricts messages between global and local controllers since only root controllers can handle OpenFlow events.
- FlowVisor (Sherwood et al., 2009) is a specialized OpenFlow controller, which acts as a transparent proxy between OpenFlow-enabled network devices and multiple controllers. FlowVisor uses switch-level virtualization to build a platform that divides a physical network into multiple logical domains.
- DISCO (Phemius et al., 2013) is an open and extensible distributed SDN control plane that is able to cope with the distributed and heterogeneous nature of modern overlay networks and wide area networks.
- ONOS (Berde et al., 2014) is an open network operating system, which supports distributed SDN control plane and provides high availability, performance and scale-out.
- Pratyaastha (Krishnamurthy et al., 2014) is an efficient elastic distributed SDN control plane and proposes a novel approach for assigning SDN switches and partition state to distributed controller instances.
- Hydra (Chang et al., 1609) is a novel SDN controller for partition and placement of SDN control plane function among different servers.
- Disttm (Hark et al., 2016) uses collaborative traffic matrix estimation to reduce the redundant measurement of flows from multiple networks dominated by multiple controllers.
- ElastiCon (Dixit et al., 2014) is an elastic multiple controllers management platform that can dynamically assign controllers according to traffic conditions. In addition, switches migration protocol and load balance algorithms are designed to make the multiple controllers networks more flexible.
- Fleet (Matsumoto et al., 2014) eliminates malicious administrator problem of multiple controllers, through threshold voting protocol and independently-created configurations.
- IRIS (Lee et al., 2014) proposes a multiple controllers architecture that adopts hierarchical SDN network to enhance scalability and availability.
- NVP controller (Koponen et al., 2014) provides a network virtualization platform that builds on SDN multiple controllers platform Onix (Berde et al., 2014).
- OpenDaylight (Controller platform, 2013) is an open source platform that supports clustering. Multiple instances of the OpenDaylight controller can work together as a entity and achieve both availability and scalability.
- PANE (Ferguson et al., 2013) provides a fine-grained allocation mechanism of network resources to multiple principals control and a solution to resolve the inevitable conflicts among multiple principals requests.
- Yanc (Guha et al., 2013) builds a file system to store network configuration and state, which makes operating network more easily.
- Smartlight (Botelho et al., 2014) is the first discussion on the design and implementation of practical fault-tolerant SDN architecture.

5. Multiple controllers design principles

Multiple controllers architecture have been deployed in many scenarios (e.g., B4 Jain et al., 2013). Oktian et al. (2017) analyze distributed SDN controller issues in different design choices (e.g., controller connection strategy, network information distribution strategy, controller coordination strategy), and those issues include scalability, failure, consistency and privacy. Moreover, we have surveyed the literature and observed three design principles that SDN with multiple controllers adopters choose during their implementation. In a more general way, three design principles are mainly focused on for consideration in this paper: consistency, fault tolerance and availability. Though these principles cannot provide comprehensive requirements to multiple controllers, they are essential factors on the multiple controllers design. This section will explain each design principle in more details.

5.1. Consistency

Distributed controllers impose potential inconsistency risks, which may cause inconsistent logic rule in switches and lead to network instability. Moreover, messages and network state exchanged among controllers may be asynchronous due to inconsistency.

Some solutions have been proposed to solve the consensus problem of multiple controllers. Bannour et al. (2017) propose a self-adaptive multi-level consistency model that helps developer implement multiple application-specific consistency models. Aslan and Matrawy (2017) use clustering techniques (sequential K-means and incremental K-means) in order to build a tunable consistency model. Muqaddas et al. (2017)

[1] Assignment between switches and controllers can also be regarded as a part of the placement problem. However, considering that traffic dynamics need to be considered for assignment problem, it is treated as an issue of multiple controllers scheduling in Section 8.2.

Table 2
Multiple controllers platforms and projects.

Name	Language	Controller Platform	Organization	Open Source	Feature
HyperFlow Tootoonchian and Ganjali (2010)	C++	NOX	University of Toronto	Yes HyperFlow source code (2013)	Suitable for large-scale networks or networks that update events occasionally
Onix Koponen et al. (2010)	Python C++ Java	–	Nicira/Google	No	Suitable for large-scale networks implementation
Kandoo Hassas Yeganeh and Ganjali (2012)	C C++ Python	–	University of Toronto	Yes Kandoo source code (2015)	More efficient in processing local events
FlowVisor Sherwood et al. (2009)	C	–	Stanford/Nicira	Yes FlowVisor source code (2012)	Switch-level virtualization
Disco Phemius et al. (2013)	Java	Floodlight	Thales Communications & Security	No	End-to-end flow management
ONOS Berde et al. (2014)	Java	Floodlight	Open Networking Laboratory	Yes ONOS source code (2014)	Scale-out, fault tolerance
Pratyaastha Krishnamurthy et al. (2014)	Java	–	University of Wisconsin Madison	No	Switch assignment, applications state storage and access
Hydra Chang et al. (1609)	Java	–	Purdue University	No	Functional slicing
DISTTM Hark et al. (2016)	Java	Floodlight	Technische Universitt Darmstadt	No	Collaborative traffic matrix estimation
ElastiCon Dixit et al. (2014)	Java	Floodlight	Purdue University	No	Switch migration, load balance algorithm
Fleet Matsumoto et al. (2014)	Python	–	Carnegie Mellon University	No	Malicious administrator problem detection and elimination
IRIS Lee et al. (2014)	Java	Floodlight	Electronics and Telecommunications Research Institute	Yes IRIS source code (2013)	Distributed control plane for carrier-grade large networks
NVP Controller Koponen et al. (2014)	Python C++	Onix	VMware	No	Network virtualization
OpenDaylight Medved et al. (2014)	Java	–	Linux Foundation	Yes OpenDayligh source code (2013)	Clustering, scalability and availability
PANE Ferguson et al. (2013)	Java	–	Brown University	No	Fine-grained allocation mechanism
Yanc Guha et al. (2013)	C++ Python	–	University of Colorado at Boulder	No	File system design
Smartlight Botelho et al. (2014)	C++ Java	Floodlight	University of Lisbon	No	Fault tolerance

research the control traffic among the controllers due to the adopted consistency protocols. Sakic et al. (2017) introduce an adaptive consistency model for SDN controllers.

How to guarantee the consistency among a plurality of controllers becomes a crucial issue of multiple controllers design. Previous studies on consistency and network performance show that inconsistency of control planes has significant effects on the network performance (Dan et al., 2012). In general, consistency of multiple controllers with SDN covers state consistency, version update consistency and rules update consistency:

- **State consistency** refers that distributed state across cluster members is replicated, which requires every controller has an identical global view.
- **Version update consistency** ensures that multiple controllers have newest state rather than hold the old state of the network.
- **Rules update consistency** requires that controllers and switches need to keep the same forwarding policies for stable forwarding.

5.1.1. State consistency

State consistency includes both strong and weak consistency (Botelho et al., 2013). Strong consistency means that when a controller updates network state, read operations of other controllers will return the newest updated values. Weak consistency means that if no new state is updated, all controllers will obtain the final update to keep the eventual state consistency, commonly called “eventual consistency”. Some existing multiple controllers platforms can achieve state consistency (see Table 3). Onix Koponen et al. (2010) uses Network Information Base (NIB) to store the global network view and supports two kinds of update and dispense mechanisms of NIB information. Replication and transactional database mode provides a reliable mechanism to guarantee strong consistency. And Distributed Hash Table (DHT) mode uses general Application Programming Interface (API) to guarantee weak consistency of network view. Onix is suitable for the situation that network events update frequently and has high availability requirements.

HyperFlow (Tootoonchian and Ganjali, 2010) transmits network state to controllers through a publish/subscribe system and implements a consistent network-wide view among multiple controllers. Different controllers share the same global network view, while each single controller just controls its own domain.

For ONOS (Berde et al., 2014), controllers send events by publish/subscribe method. Multiple communication channels are used among controllers similar to HyperFlow. ONOS provides two primitives for managing distributed state (ONOS Distributed Primitives, 2015). Cassandra (Lakshman and Malik, 2010) is used to store network information and provide weak consistency, and Raft (Agrawal, 1985) offers strong consistency.

Table 3
State consistency comparison of multiple controllers platforms.

Controller platform	Mechanisms	Consistency
HyperFlow Tootoonchian and Ganjali (2010)	Event-based	weak
Onix Koponen et al. (2010)	Replicated transactional database	strong
	DHT	weak
ONOS Berde et al. (2014)	Cassandra database	weak
	Raft	strong
DISCO Phemius et al. (2013)	Messenger, agent	strong
OpenDaylight Medved et al. (2014)	Raft	strong
ElastiCon Dixit et al. (2014)	Hazelcast	strong
SCL Panda et al. (2017)	Event logs	strong
FPC Ho et al. (2016)	Paxos	strong

DISCO (Phemius et al., 2013) sends information to other controllers using a messenger module based on four agents (i.e., Monitoring, Reachability, Connectivity, Reservation) and AMQP (Advanced Message Queuing Protocol) through publish/subscribe communication channel.

OpenDaylight (ODL) (Medved et al., 2014) provides an architecture of SDN multiple controllers, which is called ODL Clustering (Controller platform, 2013). ODL realizes the consistency by Raft algorithm (Agrawal, 1985), which periodically elects a controller as a leader and sends all data changes to the leader to handle the update (Kim et al., 2017).

ElastiCon (Dixit et al., 2014) use Hazelcast (Johns, 2013) to implement the distributed data store. Hazelcast provides strong consistency, transaction support, and event notifications. Its in-memory data storage and distributed architecture ensures both low latency data access and high availability.

Simple Coordination Layer (SCL) (Panda et al., 2017) can transform any single-image SDN controller design into a distributed SDN system. In order to guarantee the state consistency, all live controllers in SCL periodically exchange their current network state by event logs.

FPC (Ho et al., 2016) proposed a Fast Paxos-based Consensus algorithm to handle the controller consensus problem. FPC achieves consistency through three roles: Listener, Proposer and Chairman. All controllers are listeners initially. After receiving an event, a listener will become a proposer and poll other controllers whether disposing the event. If the proposer receives acceptance votes from majority of controllers, it will convert to chairman, and then handle the event and make other controllers to update.

5.1.2. Version update consistency

For the consistency in controller version update process, some studies have suggested ways to detect inconsistency. For example, Header Space Analysis (Kazemian et al., 2012) and VeriFlow (Khurshid et al., 2012) can verify current network configuration and detect controller inconsistency through checking the data-plane. In addition to inconsistency detection, some solutions are proposed to solve inconsistency. OFRewind (Wundsam et al., 2011) records a temporally consistent trace of the controller domain and replays network events using a hypervisor. But it is not concerned with recreating state in a correct manner. HotSwap (Vanbever et al., 2013) achieves no interrupted controller upgrade by recording historical network events in the old controllers and duplicating on new controllers. Perešíni et al. (2013) make use of multi-commits transactional semantics to realize a consistent message processing.

5.1.3. Rules update consistency

The original and updated rules should both be consistency, but disseminating the rules update is an inherently asynchronous process, which may cause potentially inconsistent state (Förster et al., 2016). A typical case is that, when controller transfers new flow entries to all switches, these updates are applied asynchronously. Such asynchronous operations may lead to some switches have been updated many times, while others still have not been update (Panda et al., 2017). How to guarantee the consistency of rules update is a great challenge. Reitblatt et al. (2011) propose SDN consistency of rules update for the first time. They introduce packet consistency and flow consistency to avoid the abnormal state in the update process. On this basis, rules update theory model is put forward to illustrate the correctness of update policy (Reitblatt et al., 2012). Mcgeer (2012) presents safety rules update protocol based on OpenFlow, guaranteeing the consistency of packet rules while reducing the occupation of forwarding entries in the update process. Canini et al. (2015) propose FixTag, a wait-free algorithm to address consistency problem of rules update via a transactional interface.

Table 4
Consistency strategies.

Consistency types	Consistency strategies
State consistency	publish/subscribe system Tootoonchian and Ganjali (2010), Berde et al. (2014), Phemius et al. (2013), consistency algorithm (e.g., Raft Controller platform, 2013,Paxos Ho et al., 2016)
Consistency in controllers version update	Record in old controllers, playback in new controllers Vanbever et al. (2013) Transactional semantics Perešíni et al. (2013)
Consistency in rules update	Packet consistency and flow consistency, rules update theory model Reitblatt et al. (2012) Safety rules update protocol based on OpenFlow Mcgeer (2012)

5.1.4. Remarks on consistency

Table 4 summarizes all three types of consistency strategies. In practice, it is difficult to achieve all kinds of network consistency for all the time in all situations. Consistency strategies are put forward according to different situations, such as state consistency, version update consistency and rules update consistency. These strategies offer references when facing different network situations. Appropriate strategies can be chosen according to actual conditions. For example, Onix can be used to provide consistency model to ensure the consistency of the network view in the control plane. Then, recording historical network events in old controllers and duplicating in the new controllers can be used to keep consistency in controllers version update. Safety rules update protocol can be used to avoid loop, black hole and security problems due to inconsistent switch forwarding rules during the rules update process.

5.2. Fault tolerance

Fault tolerance refers to the fact that the network can still maintain normal operations when network faults happen, e.g., primary controller fails (Schiff et al., 2016). Fault tolerance is a preventive measure to improve controller reliability, ensuring the safe operation and high performance of networks. Similar to traditional networks, achieving fault-tolerant communication is one of the important goals in controller design (Botelho et al., 2013). However, unlike traditional networks, fault tolerance is not only associated with the resilience of data plane, but also related to distributed control plane in SDN (Sridharan et al., 2017). Table 5 summarizes the fault tolerant mechanism of existing multiple controllers platforms.

Using multiple controllers can effectively alleviate the impact of controller faults. Fault tolerance can be achieved by carefully determining the placement of controllers (see Section 7.2 for more details of controller placement). Current researches mainly improve fault tolerance through passive or active replication mechanisms (Fonseca et al., 2012, 2013).

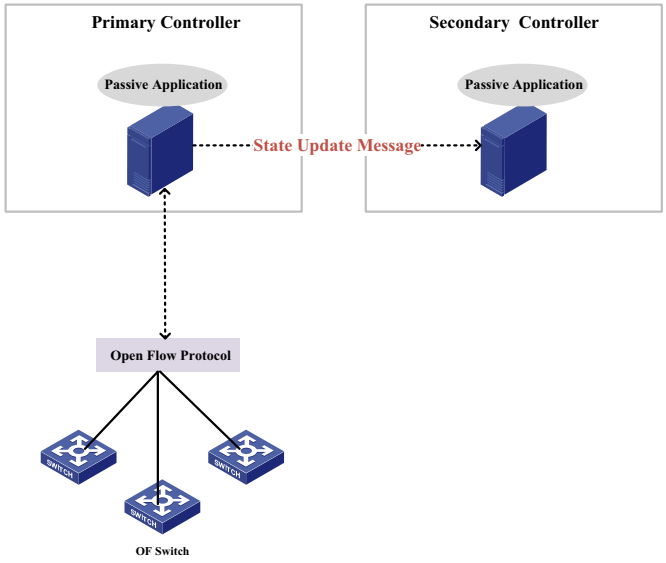

Fig. 3. Passive replication.

- **Passive Replication.** Each switch can establish communication with only one controller (primary controller). When the primary controller enters in a failure state, one of the backup controllers will be chosen to take control of the network. This approach is also known as Primary-Backup Replication (see Fig. 3).
- **Active Replication.** It is another well-known replication technique. In this approach, switches can keep connection with multiple controllers simultaneously, and when one controller fails, others can still control switches without switching (see Fig. 4).

The forwarding of switches is completely controlled by controllers. When a network node or link fault occurs, it not only affects the data flow, but also causes flow interruption between controllers and switches. The direct approach to solve this problem is failover mechanism (Sharma et al., 2013). A switch sends a message to the controller when fault is detected by the switch. Afterwards, the controller will recalculate according to the message or choose an available route in the backup route list that is calculated in advance, and then configure the relevant switches. To meet operators' demands for fault tolerance, protection mechanism is needed to ease the negative impact of separating control plane and data plane. Beheshti and Zhang (2012) propose algorithms for improving the resiliency by maximizing

Table 5
Fault tolerance comparison of multiple controllers platforms.

Controller platform	Failure type	Solution
HyperFlow Tootoonchian and Ganjali (2010)	Controller failure	Nearby controllers serve as a hot standby
Onix Koponen et al. (2010)	Link or switch failure	Backup paths
	Onix instances failure	Active replication
ONOS Berde et al. (2014)	ONOS instances failure	Redundant instances
Hydra Chang et al. (1609)	Controller failure	Replication of controllers configuration
ElastiCon Dixit et al. (2014)	Controller failure	Dynamic controller migration
Fleet Matsumoto et al. (2014)	Link failure	Switching to a workaround path for the failed link
IRIS Lee et al. (2014)	Controller failure	Controller switching
NVP Koponen et al. (2014)	Controller failure	Standby controllers
PANE Ferguson et al. (2013)	Link or switch failure	Forwarding policy reconfiguration
SmartLight Botelho et al. (2014)	Controller failure	Replicated shared database for recovery

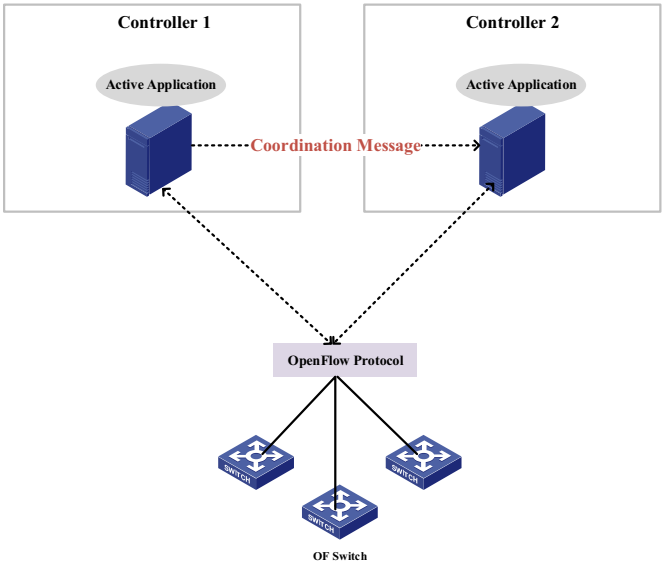

Fig. 4. Active replication.

the possibility of fast failover, which is achieved through resilience-aware controller placement and control-traffic routing in the network. Kempf et al. (2012) adopt monitoring function and protection mechanism to manage the OpenFlow network faults. In addition, segment protection mechanism can also be used to avoid controllers' participation in recovery process (Sgambelluri et al., 2013).

In addition, modifying packets to make them carry backup route or fault message is another way to improve fault tolerance of SDN. SlickFlow (Ramos et al., 2013) adds backup route message into packet headers. When faults happen, packets will be sent through the fault-free route according to the backup route message. INFLEX (Arajo et al., 2014) uses tags to represent virtual forwarding message. Exporting switches modify the tags to change the forwarding policies. Borokhovich et al. (2014) propose tags and fast failover algorithm based on graph theory to improve fault tolerance.

Different mechanisms can be applied to solve different network faults. Every mechanism has its own advantages and disadvantages. Table 6 summarizes some typical mechanisms and features for fault tolerance. Additionally, more recently papers are listed in the table. Failover mechanism is the most common approach. However, this reactive mechanism not only brings additional computational burden to controllers, but also requires a long recovery time, due to the interaction between switches and controllers. These mechanisms can be divided into preventive protection mechanisms and recovery after fault mechanisms. Preventive mechanisms have short recover time by backing up route or common fault messages. Recovery after fault mechanisms may cause computation burden and long recovery time. These two mechanisms can be combined to improve fault tolerance.

5.3. Availability

Availability refers to the proportion of time an SDN system is in a functioning condition (Nencioni et al., 2017). Indeterminacy among multiple controllers may cause potential risks and dangers, which reduce the availability. Three factors can be considered to improve controller availability: rules backup, controller load and switch requests.

Rules backup can improve the availability of controllers. RuleBricks (Williams and Jamjoom, 2013) proposes a high availability solution aiming at rules backup. In RuleBricks, different rules correspond to different colors of "bricks". Among them, the top "bricks" correspond to current active rules. If the network node fault result in disappearance of a certain color "bricks", the following backup "bricks" will emerge as new active rules. Through setting "bricks" and optimizing selection and operation, RuleBricks can effectively limit flow redistribution and rules explosion.

Controllers need to deal with a large number of requests from switches, but overweight load might affect the availability (Toumi et al., 2016). Distributed architecture can be used to load balance, especially in the hierarchical architecture. However, uneven distribution of network traffic would reduce availability. In light of this problem, ElastiCon (Dixit et al., 2013) dynamically adjusts the traffic among the controllers by periodically checking load window. When the load window changes, it will dynamically expanse or compress controller pool to adapt to the current demand. If the load exceeds maximum of controller pool, new controllers would be added to ensure the availability. Some multiple controllers platforms, Onix (Koponen et al., 2010), Fleet (Matsumoto et al., 2014) and IRIS (Lee et al., 2014) also use controller load balancing to guarantee high availability.

Reducing the number of switches requests can also improve the availability of the control layer. In DIFNAE (Yu et al., 2011), data plane are allowed to process flow while controllers just classify rules and send the rules to the switches. Therefore, DIFANE can adapt to the large-scale network topology and handle more forwarding rules. ParaFlow (Song et al., 2017b) is also a fine-gained parallel SDN controller, which can adapt to large-scale network. DevoFlow (Curtis et al., 2011) divides flow into short flow and large flow, then sets up some different rules in the switches according to different flow. So data plane can deal with short flow directly, and only a small number of large flow are handled by controllers. Therefore, DevoFlow can reduce the controller load in maximum and improve the availability of the controllers. Table 7 summarizes these methods that improve controllers availability.

Actually, both reducing the number of switches requests and dynamically adjusting the traffic can improve availability by adjusting controller load. The former reduces controller load by migrating the load to the switches. The latter increases the maximum of controller load according to the demand.

Furthermore, security is also important to availability. If multiple controllers are in deplorably insecure conditions, the SDN network will fall into danger, let alone the availability of multiple controllers.

Table 6
Mechanisms and the features of fault tolerance.

Mechanism of fault tolerance	Fault type	Features
Controller Replication Fonseca et al. (2012), Fonseca et al. (2013)	Controller fault	Ensure the communication between switches and controllers without interruption
Failure recovery Sharma et al. (2013), Zhang et al. (2016)	Node or link fault	Switches recovery without controllers, recovery time within 50 ms
Protection switching Kempf et al. (2012)	Node or link fault	Monitoring function, fault recovery time within 50 ms
Segment Protection Sgambelluri et al. (2013)	Node or link fault	Avoid controllers' participation in recovery process, recovery time below 64 ms
Fast Failover Beheshti and Zhang (2012), Raeisi and Giorgetti (2016), Ahmed et al. (2016)	Node or link fault	Maximize network resiliency
Packets Modification Ramos et al. (2013), Arajo et al. (2014), Borokhovich et al. (2014), Chen et al. (2016)	Node or link fault	Carry backup route or fault message to change the forwarding policies

Table 7
Methods to improve controller availability.

Methods	Examples
Rule backup	RuleBricks Williams and Jamjoom (2013)
Controller load balancing	Onix Koponen et al. (2010), ElastiCon Dixit et al. (2013), Fleet Matsumoto et al. (2014), IRIS Lee et al. (2014)
Proactive rule setup	DIFANE Yu et al. (2011),DevoFlow Curtis et al. (2011), ParaFlow Song et al. (2017b)

Therefore, security of controllers should be considered to improve availability besides the above three aspects. If security problem cannot be solved, SDN might lose the availability when deployed in large scale (Qi et al., 2016; Dacier et al., 2017).

5.4. Summary

Several design principles are discussed to enhance the architecture of multiple controllers. Different consistency requirements need different consistency strategies. Passive replication and active replication are two main mechanisms to improve fault tolerance. Rules backup, controller load and switch request reduction are methods to improve controller availability. These design principles are not independent. All principles must be taken into account in practice. Yet not all existing multiple controllers platforms meet the design principles mentioned above, some platforms only satisfy one or two principles.

6. Multiple controllers architectures

According to how multiple controllers work together, SDN multiple controllers architecture generally falls into two categories: centralized architecture and distributed architecture. Centralized architecture is considered to be the closest to the initial design of SDN, which just deploy a single controller (Kreutz et al., 2015). Distributed architecture is to deviate from the fist tendency of SDN, by using multiple controllers to form a distributed control plane (Davie et al., 2017). Besides, J. Hu et al. (2014); Y. Hu et al. (2014) introduce four kinds of controller architectures similar to the above architectures: centralized architecture, horizontal architecture that each controller has a global network topology, horizontal architecture that each controller only has a local network topology, and hierarchical architecture. Horizontal architecture can be further divided into two types considering whether each controller has global view. In this section, we mainly focus on distributed architecture with two categories, horizontal architecture and hierarchical architecture, as well as communications among multiple controllers.

6.1. Distributed architecture

Distributed architecture is an effective way to improve the scalability of controllers (Y. Hu et al., 2014; J. Hu et al., 2014; Gray et al., 2016). In such architecture, the control plane may consist of multiple controllers rather than a single controller. These multiple controllers in charge of managing different administrative domains of the network and exchanging local information with adjacent domains to enhance global policies implementation. In general, distributed controllers can use two patterns to extend, horizontal architecture and hierarchical architecture respectively (Gray et al., 2016).

6.1.1. Horizontal architecture

For horizontal architecture (see Fig. 5), all controllers are put into disjoint domains, which is also called flat model. Controllers manage their own network separately with equal state and communicate with each other through east-west interfaces (Benamrane and Benaini et al., 2017; Lam et al., 2016). Horizontal architecture requires that all

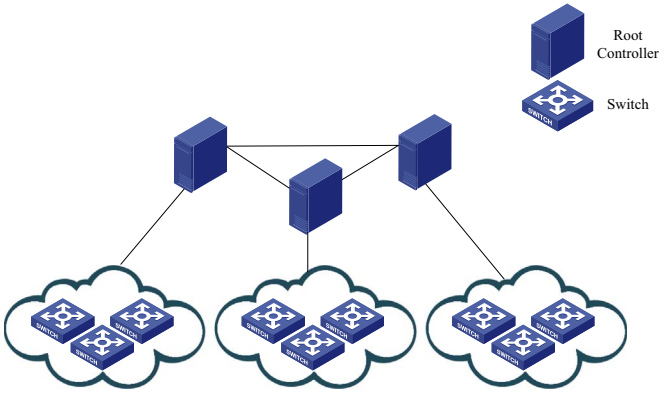

Fig. 5. Horizontal architecture.

controllers are in the same level. Although each controller locates in different domains, all controllers are logically regarded as root controllers, mastering the whole network state. When the network topology changes, all controllers will upgrade synchronously without complicated operations, only by adjusting the mapping to the address of controller. Therefore, such architecture has limited influence on data plane. Onix (Koponen et al., 2010), HyperFlow (Tootoonchian and Ganjali, 2010) and FlowVisor (Sherwood et al., 2009) are the typical examples of horizontal architecture. However, horizontal architecture still has some disadvantages. Each controller is regarded as root controller to master the whole network state, while it just controls the local network. This will cause resource waste and increase entire load of controllers when updating the network, due to the interaction among controllers. Moreover, different domains belong to the operators who own different economic entities, making it difficult to ensure equal communication among controllers in different domains.

6.1.2. Hierarchical architecture

Controllers can also be managed in a hierarchical architecture (Koshibe et al., 2014), see Fig. 6. Hierarchical architecture classifies all controllers into root controllers and local controllers according to their purpose. Local controllers are close to switches relatively and responsible for the network state in their own domain. Root controllers are responsible for maintaining the whole network information. Local controllers must inquiry root controllers before handling the inter-domain events, similar to client and server model (Oktian et al., 2017).

IRIS (Lee et al., 2014) and Kandoo (Yeganeh and Ganjali, 2012) are examples of distributed SDN controller project using hierarchical model. Kandoo presents a method to exchange of information between switches

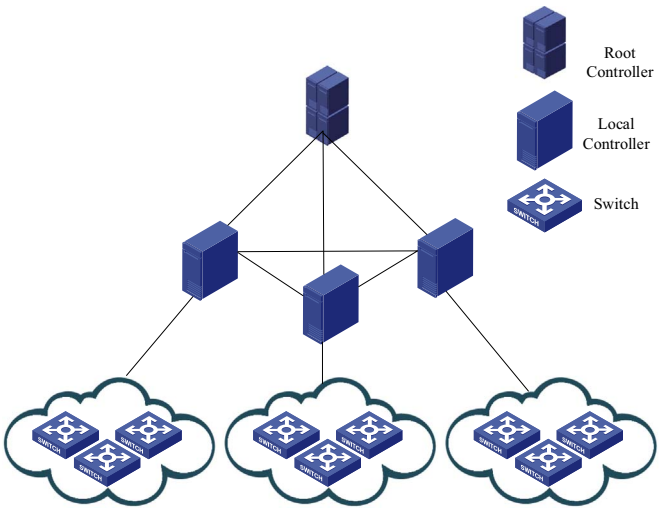

Fig. 6. Hierarchical architecture.

Table 8
Advantages and disadvantages of different multiple controllers architectures.

Controller architecture	Advantages	Disadvantages	Examples
Horizontal architecture	Upgrade synchronously without complicated operations	Resource wasting, increase controller load	Onix, HyperFlow, ONOS, DISCO FlowVisor
Hierarchical architecture	Reduce load, fast response	May increase controller number	Kandoo,IRIS

and controllers. When a switch forwards the packet, it will inquiry its nearby local controller. If this packet belongs to local message, the local controller will send response immediately. Otherwise, the local controller will inquiry root controller, and then forward the obtained message to the switches. In this way, it avoids frequent interaction among root controllers and reduces flow load efficiently.

6.1.3. Remarks on distributed architecture

Comparing with horizontal architecture, hierarchical architecture can reduce the network load and resource waste, which also has a faster response. But it may increase the total number of controllers in the network. From the comparison of the scalability on these two architectures, horizontal structure has similar scalability to hierarchical architecture when the number of network hosts is far greater than the number of controllers. Moreover, the scalability will be reduced with the increase of distance between the controllers. Table 8 shows the advantages and disadvantages of different multiple controllers architectures.

6.2. Controllers-to-controllers communication

East-west interface (Lam et al., 2015) is used for the communication among controllers. OpenDayLight (Medved et al., 2014) and ONOS (Berde et al., 2014) make use of clustering technology to achieve the communication among multiple controllers. East-west interface is one of the main solutions to solve the problem of control plane performance. HUAWEI proposed SDNi (Yin et al., 2012) for the interface between SDN domains to exchange information between the domain SDN controllers. SDNi is deployed in OpenDayLight, allowing the inter-domain communication among multiple controllers. However, existing distributed controllers only support the operations among the same type of controllers. With the scale enlargement of SDN deployment, the researchers begin to consider how to realize the communication among different kinds of controllers (Yu et al., 2015). At present, the researches on the east-west interface of SDN are still in the primary stage, and industry standards are still missing. The standards of east-west interface should be decoupled with the SDN controllers to realize the communication among the controllers of different factories.

West-East Bridge (WEBridge) (Lin et al., 2015) is an interaction mechanism for the inter-domain communication of multiple controllers. Each controller has a WE-bridge module and they establish neighbor relationship through west-east bound bridge, which is used to exchange inter-domain network view among the controllers (see Fig. 7). WEBridge adopts publish/subscribe pattern as information interaction mechanism, and implements the protocol referring to Border Gateway Protocol (BGP). Five types of messages are defined, including open, update, notification, keepalive and viewerfresh. WEBridge has already been deployed in practical networks such as China Education and Research Network (CERNET), Internet2 and China Science and Technology Network (CSTNET). Juniper Company supports east-west interface for OpenContrail controller (Guan et al., 2013). Different from WEBridge, OpenContrail controllers regard Inter Border Gateway Protocol (iBGP) as east-west interface. Different controllers exchange data through iBGP to realize synchronization among multiple controllers. In addition, in order to improve the extensibility of control plane, BGP routing reflection can be used to aggregate states of controllers.

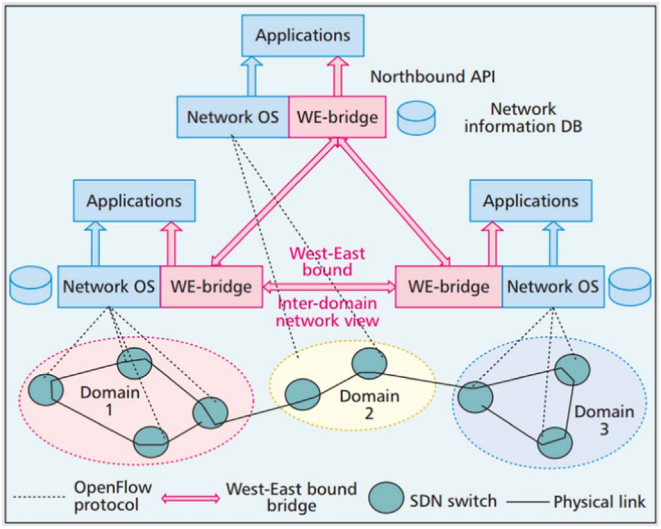

Fig. 7. West-east bridge (Lin et al., 2015).

6.3. Summary

This section analyzes the architecture of multiple controllers. Distributed architecture is generally divided into horizontal mode and hierarchical mode. Moreover, this section elaborates the communication among controllers that is also known as east-west interface. SDNi and WEBrige both provide solutions of east-west interface, and the latter one has already been deployed in real networks. However, there is no standard of east-west interface yet.

7. Multiple controllers placement

Distributed controllers need to act as a logically centralized control plane (Xiao et al., 2014). This means sharing information among controllers is needed to keep a global consistent view of the network and exchange data over the control plane. During this process, one critical problem is how to choose the number of controllers and appropriate locations of these controllers.[2] Many metrics have been proposed to determine multiple controllers placement, e.g., propagation latency, network reliability and controller load. Some works focus on a certain metric, while others integrate several metrics at the same time to find an optimal trade-off (Sallahi and Sthilaire, 2015; Tuncer et al., 2015).

7.1. Propagation latency

The propagation latency from switches to controllers influence the response speed of controllers to the network events (Liao et al., 2017). Thus, propagation latency is an important metric of controller placement scheme to determine the number of controllers and their locations.

Heller et al. (2012) compare different placement ways with different numbers of controllers according to average or worst propagation

[2] As explained in Section 3.2, considering network dynamics for assignment between controllers and switches, it is discussed as one scheduling problem in Section 8.

latency from switches to controllers using K-median and the K-center algorithms. If randomly choosing a placement with k controllers, the average latency is between 1.4x and 1.7x larger than the optimal placement, and the worst-case latency is between 1.4x and 2.5x larger than the optimal placement. They point out that there does not exist an universal rule of controller placement, but there is a guarantee for finding optimal trade-off with respect to latency.

An optimal scheme with propagation latency is proposed in Heller et al. (2012). However, it does not consider the weight of the nodes and regards them as the same. The weight of nodes in the real network is uneven (Hock et al., 2014), so these results are inapplicable in practical network. Yao et al. (2015) consider various nodes weight when placing the controllers. He et al. (2017) focus on controller placement for average flow setup time improvement in SDN, by building a controller placement model to optimize minimum average flow setup time with respect to different traffic conditions. Zhao and Wu (2017) provide a scalable placement architecture to reduce the link delay through integer linear program and heuristic algorithm. In Zhang et al. (2017), the authors discuss Pareto-optimal controller placement considering controller-to-switch and controller-to-controller delays for wide area network topology. Both the latency from switches to controllers and weight of switch nodes should be considered simultaneously. Additionally, the propagation latency between multiple controllers should be taken into account in the distributed architecture.

7.2. Reliability

Network failures may result in the disconnection between controllers and switches and failures in data plane (Yao et al., 2013). In order to improve fault tolerance of SDN, the reliability and resilience (Scott-Hayward, 2015) of SDN should be considered when deploying multiple controllers (J. Hu et al., 2014; Y. Hu et al., 2014).

Hu et al. (2013) introduce expected percentage of control path loss to reflect the reliability, where the control path loss is the number of broken control paths due to network failures. Random placement, l-w-greedy and simulated annealing are implemented. They found that simulated annealing algorithm has the best reliability. The deployments with best reliability in different topologies have similar results. With the number of controllers increasing, the reliability goes up at first and then goes down. The study indicates that appropriate number of controllers is between $0.035n$ and $0.117n$ (n is the number of nodes).

The fault tolerant controller placement problem is introduced by Ros and Ruiz (2014). They develop a heuristic algorithm that computes placements based on reliability, evaluating the algorithm on 124 publicly available network topologies. Results show that if each node connects to two or three controllers, it can provide more than five nines reliability. In addition, the number of controllers is related to the network topology rather than the network size, and, normally, 10 controllers are enough.

Zhang et al. (2011) consider failures of controller and network elements when deploying controllers. Nodes that are unable to connect to the controller due to failure are called lost nodes, and the number of lost nodes is optimized as the optimization target. A min-cut based controller placement algorithm, which improves the reliability, is proposed and compared with the greedy based algorithm.

It is obvious that too few controllers would decrease reliability and performance, while too much controllers increases overall cost. To get a placement with best reliability, the network topology and the network scale are needed to be deliberated.

7.3. Controller load

In addition to the deployment schemes based on propagation latency and reliability, overload of controllers is another important metric for placement of multiple controllers.

Yao et al. (2014) define the controller deployment problem with load constraint and design the deployment algorithm to reduce the number of required controllers and load. As the results shown, Capacitated K-center algorithm can avoid overload by reducing the number of controllers. As the number of controllers increases while no overload happens, 5 controllers are needed with capacitated K-center algorithm for 60% topologies, and 15 controllers are needed for all the topologies while 40 controllers are needed with K-center algorithm.

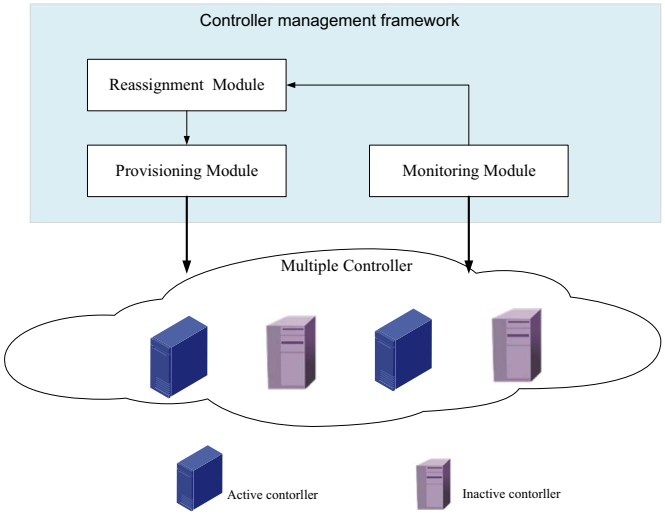

Fig. 8. Multiple controllers management framework.

Bari et al. (2013) propose a heuristic algorithm to solve the dynamic controller provisioning problem through Integer Linear Programming (ILP), which reduces flow setup times and controller load by dynamically changing the quantity and location of controllers. A controller management framework is proposed (see Fig. 8). Monitoring module monitors controllers to ensure aliveness and collects relevant statistics from them. When controllers failure and inactive, reassignment module will be triggered by the monitoring module, according to the collected statistics. And then, provisioning module provisions controllers as required and changes switch to controller associations.

Rath et al. (2014) propose a non-zero-sum game theory to work out a simple and low-complexity algorithm that can distribute load on all controllers uniformly. It can be applied as an optimization engine at each SDN controller to compute a payoff function. The engine compares its own payoff value with its neighbors and takes appropriate decisions whether new controllers should be added, or existing controllers should be deleted.

Different from K-median and K-center algorithm, these algorithms mentioned above are implemented in the network with dynamic traffic. So the experiment results are much closer to the real network. They reduce the controller load and avoid overload by adjusting the number and location of controllers dynamically. The results show that these algorithms can reduce controller load in various degrees. As the increasing of the network scale, controllers load must become heavier. It is necessary to find an effective algorithm that is suitable to such heavy load.

7.4. Multiple metrics

Multiple metrics may be considered together when deploying controllers. However, in most scenarios, an optimal controller deployment scheme may not exist. It is needed to find a trade-off among those multiple metrics.

Jimenez et al. (2014) consider the reliability besides controller load and propose a placement scheme using K-critical algorithm to find the minimum quantity and location of controllers in order to meet the reliability and load.

Hock et al. (2014) consider various metrics including network reliability, controller load and propagation latency and use Pareto-

Table 9
Deployment algorithms according to different metrics.

Deployment algorithm	Propagation latency	Reliability	Controller load
K-median and the K-center algorithm Heller et al. (2012)	√		
Random placement, l-w-greedy and simulated annealing Algorithms Hu et al. (2013)		√	
Heuristic algorithm for fault tolerant controller placement Ros and Ruiz (2014)		√	
Min-cut based controller placement algorithm Zhang et al. (2011)		√	
Heuristic algorithms for dynamic controller provisioning problem Bari et al. (2013)			√
Algorithm based on non-zero-sum game theory Rath et al. (2014)			√
Capacitated K-center algorithm Yao et al. (2014)			√
K-critical algorithm Jimenez et al. (2014)		√	√
Algorithms for Pareto-based Optimal Controller placement with POCO framework Hock et al. (2014), Lange et al. (2015)	√	√	√

based Optimal COntroller-placement (POCO) (Hock et al., 2013) framework to compute resilient placements.

Similar to Hock et al. (2014), Lange et al. (2015) point out that some metrics including controller-switches latency, controller-controller latency, controller load and network failures play an important role in determining the number and location of required controllers in large-scale network. Algorithms for Pareto-based Optimal Controller placement are computed with POCO framework to find a optimized trade-off among multiple metrics.

Table 9 summarizes these placement algorithms according to different metrics. When multiple metrics come into consideration, it is difficult to find a trade-off. One scheme may reduce propagation latency and controller load while have a little weak reliability. Another scheme may have short propagation latency and strong reliability while have a little heavy load. It is a thorny problem which one is a balanced trade-off. Hence, it is important to determine which metric influence the network greatly and choose the optimal trade-off.

7.5. Summary

This section discusses the placement problem of multiple controllers. Firstly, three single metrics are presented to characterize the placement of SDN multiple controller, including propagation latency, reliability and controller load. Then multiple metrics scheme can be used to find an optimized trade-off, which make a better solution to multiple controllers placement. It's difficult to choose which metric or solution is the best one, but the most optimal placement can be implemented, according to concrete requirements.

8. Multiple controllers scheduling

Although multiple controllers solve the problem of single point of failure, efficient scheduling of multiple controllers is still a great challenge. Efficient scheduling policies or algorithms can enhance the network performance (Thakur and Goraya, 2017). Hence, in this section, four issues of multiple controllers scheduling are discussed: decision manner, dynamic controllers assignment, migration and load balance.

8.1. Centralized decision and distributed decision

At present, most distributed controller architectures concentrate on how to achieve logically centralized network view for multiple controllers. However, some architectures propose static mapping between switches and controllers. Static mapping means that controllers control specified switches fixedly, meanwhile, control and anti-control relationship can not be changed. In real networks, the network traffic continues to change dynamically with time and space. If mapping relationship is static, when a great change of the traffic happens on a certain switch, its assigned controller may receive many flow setup requests in a short time, which causes controller overload. To realize load balance of multiple controllers, a dynamical mapping relationship should be put forward to make switches requests distribute evenly to each controller. Control and anti-control relationship can be changed according to dynamic traffic.

There are two ways to make such decisions: centralized decision and distributed decision. The structures of both centralized and distributed decision are shown in Fig. 9.

In centralized decision, a super controller is used to balance the load of all controllers, while other controllers act as ordinary controllers. The super controller collects the global load information from other controllers and makes decisions about load balance. When the super controller detects the imbalance of controllers, it will send the commands to the controllers to conduct switches migration.

For distributed decision, all controllers act as both ordinary controllers and super controllers, which means that each controller can collect other controllers load information and make decisions directly without the commands from the super controller.

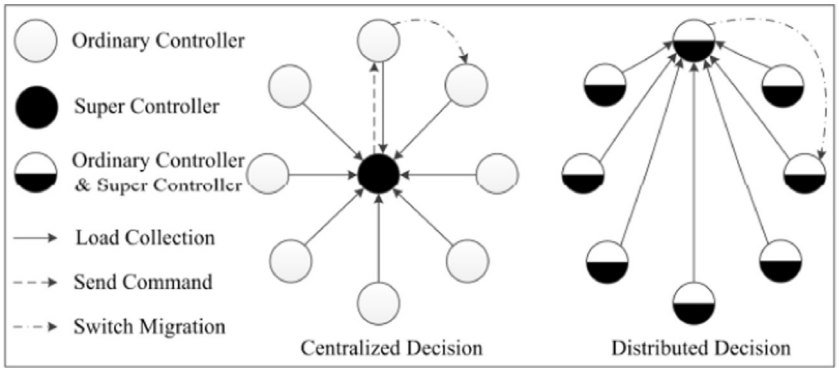

Fig. 9. Centralized decision and distributed decision (Zhou et al., 2014).

Compared with distributed decision, centralized decision may increase the latency and overload of the super controller, due to the super controller needs to exchange information with the other controllers periodically and frequently. Furthermore, if the super controller encounters a fault (e.g., it cannot make decisions or send commands), the load balance mechanism will fail.

8.2. Dynamic controller assignment

Multiple controllers are used to improve the scalability and robustness. Nevertheless, static assignment between switches and controllers may cause load imbalance among the controllers, which motivates dynamic controller assignment. The purpose of dynamic assignment between switches and controllers is to minimize both the controllers response time and the amount of inter-controller load.

Wang et al. (2016) formulate the dynamic controller assignment problem as a stable matching problem with transfers, and propose a hierarchically two-phase algorithm as solution. Pratyaastha (Krishnamurthy et al., 2014) regards the controllers assignment problem as a variant of the multi-dimensional bin packing problem. They formulate the assignment problem as an integer linear program (ILP), and then propose a heuristic approach to solve it. Bari et al. (2013) present a management framework to reduce the flow setup delay and communication overload by periodically reassigning switches to controllers.

The studies show that dynamic assignment yields a 86% decrease in controllers response time (Wang et al., 2016) and a 42% reduction in controllers operating costs (Krishnamurthy et al., 2014). Dynamic controllers assignment can significantly improve the performance in multiple controller scheduling.

8.3. Migration mechanisms and load balance algorithms

Flow setup requests in different switches should be dynamically allocated to appropriate controllers. Migration mechanism means that when network flow changes, overloaded controllers should be able to transfer the load to other controllers (Cheng et al., 2015), so as to improve the overall capability of control plane and provide rapid response of switch requests. Specially, migration mechanisms can also improve controllers availability.

Serializability of messages should be considered in switch migration mechanisms. For example, when migrating switch from controller A to controller B, it needs to ensure that controller A and B do not respond the same switch message simultaneously. Moreover, migration of traffic may cause suddenly change of switch requests and network delay (Koohanestani et al., 2017), leading to an uncertain order of messages arriving at controller A and B. Ignoring the order of message processing on both controllers may cause inconsistency between the control plane network view and the real network state.

Some migration mechanisms have been proposed as an effective solution of adjusting load balance. BalanceFlow (Hu et al., 2012) uses switch migration mechanism based on centralized decision. When controllers suffer uneven load, the super controller runs load balance algorithm to redistribute the mapping between the controllers and switches to realize the load balance of the controller. BalCon (Cello et al., 2017) is an algorithmic solution designed to tackle and reduce the load imbalance among the SDN controllers through proper SDN switch migrations. Wang et al. (2017) make an efficiency switch migration scheme for load balancing in SDN controllers. Song et al. (2017a) use a low-cost flow-stealing method to build a lightweight load balance platform for SDN multiple controllers. SCPLBS (Zhong et al., 2017) proposes a cooperative platform for load balancing and failure recovery.

Zhou et al. (2014) present DALB, a dynamic and adaptive load balance algorithm based on distributed decision, running as a module of SDN controller. This module consists of load measurement, decision maker and load collection. Each controller can measure and collect its own load and other controllers load to determine when migrations are needed. If controller load exceeds a certain threshold, controllers will make decision and migrate elected switches to balance the load of all controllers. Yu et al. (2016) propose a load balancing mechanism based on a load informing strategy for multiple distributed controllers. By using distributed decision, the scheme is designed as a module of SDN controller with four components:load measurement, load informing, balance decision and switch migration. Cheng et al. (2015) also design a distributed algorithm for switch migration problem.

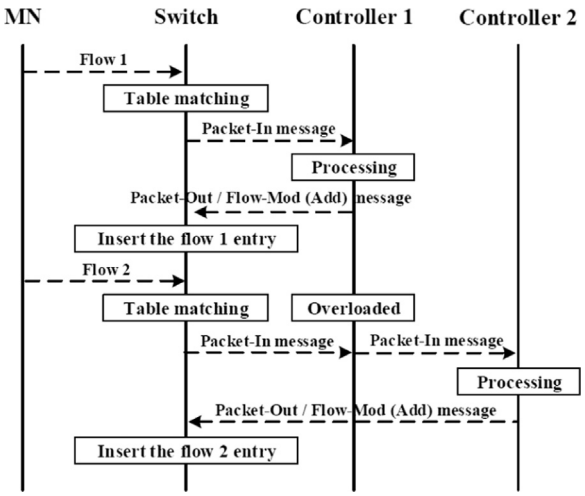

Fig. 10. Flow migration mechanism (Kyung et al., 2015).

Yao et al. (2015) propose a dynamic switch migration algorithm adjusting the dynamic flow to realize controller load balance in multiple SDN domains. Ye et al. (2017) introduce a synthesizing distributed algorithm to solve switch migration problem. Kyung et al. (2015) introduce a load distribution scheme on a per-flow basis. Fig. 10 shows the flow migration mechanism. When a switch receives a flow from Mobile Node (MN) that is not matched to any existing flow entries, then the switch will send a Packet_In message to controller 1. If controller 1 is not overloaded, it will process the message and respond a Packet_Out/Flow_Mod (Add) messages to the switch. When the controller load reaches a threshold, the controller 1 will migrates the messages to other controller(e.g., controller 2) that is not overload to handle the messages. Numerical results point out that this scheme has high controller capacity utilization ratio.

8.4. Summary

After deploying the multiple controllers, maintaining the stabilization of multiple controllers is significant. This section elaborates the scheduling of multiple controllers. Several scheduling measures are deliberated. Centralized decision and distributed decision are two kinds of way to realize controllers load balance. Dynamic controllers assignment provides a reduction in controllers response time and controllers communication overload. Migration mechanisms and load balance algorithms are regarded as failover methods to enhance availability.

9. Future research issues

There are still some problems in multiple controllers research progress. How to solve these problems and realize further development and optimization of SDN are the main research directions of SDN with multiple controllers (Wang and Matta, 2014; Akyildiz et al., 2016). Several potential future research issues on SDN with multiple controllers are summarized as below.

(1) **Coordination between multiple controllers and switches.** When SDN is deployed in large-scale networks, they may consist of

multiple domain with multiple controllers or different administrative vendors (Wang et al., 2016). Different operators may implement different policies in the data plane, which may result in conflict. Hence, how to coordinate different policies and avoid conflict are critical issues.

(2) **Standardization protocols in east-west control plane** There is no industry recognized standardized east-west protocol for distributed control plane, which makes the development of SDN hampered when it comes to large-scale network environments (Yu et al., 2015). Developing east-west protocol requires to guarantee the consistency state among heterogeneous controllers and provide high performance in latency and load. Therefore, east-west protocol development will be one of the important problems to be solved.

(3) **Standardized northbound interface development.** SDN provides open source northbound interfaces for network applications developing. However, there are many kinds of controllers with different development languages and northbound interfaces. It is difficult to extend management interfaces in large-scale networks with multiple controllers. In consequence, developing a standardized northbound interface to hide the diversity of multiple controllers is a desiderate task at present.

(4) **Dynamic load balancing mechanism for multiple controllers.** Deploying multiple controllers in a large-scale SDN network need to considerate traffic engineering, an important subject for network performance optimization. SDN traffic engineering demands a dynamic load balancing mechanism that adapts burst traffic and adjusts controllers load dynamically. Effective traffic balancing solution still remain to be further studied.

(5) **Integrating network virtualization and SDN with multiple controllers.** Network virtualization proposes an abstraction from underlying infrastructure which can adjust workloads and resources flexibly in order to meet diverse service requirements (Chowdhury and Boutaba, 2010). Integrating the SDN and NFV may trigger innovative designs that fully exploit the advantages of both paradigms (Duan et al., 2016). However, current research focuses on the virtualization infrastructure. How to implement virtualization on multiple SDN controllers is an important research topic that deserves thorough investigation.

10. Conclusions

SDN improves the network utilization efficiency and realizes the programmability. However, a single controller restricts the scalability and reliability of SDN networks. Multiple controllers architecture is needed in future development of SDN because of its high scalability and availability. In this paper, a comprehensive survey on recent progress of multiple controllers with SDN is conducted. Several existing aspects for multiple controllers such as design principles, architectures, placement and scheduling are introduced and summarized. Also, the strengths and weaknesses of existing relevant research results are analyzed.

Acknowledgements

This work is partially supported by Chinese National Research Fund (NSFC) No. 61772235 and 61402200; the Fundamental Research Funds for the Central Universities (21617409); the Opening Project of Guangdong Province Key Laboratory of Big Data Analysis and Processing (2017009).

References

Agrawal, P., Raft: A recursive algorithm for fault tolerance. In: ICPP, 1985, pp. 814–821.
Ahmed, R., Alfaki, E., Nawari, M., 2016. Fast failure detection and recovery mechanism for dynamic networks using software-defined networking. In: Proceedings of Conference of Basic Sciences and Engineering Studies (SGCAC), IEEE, pp. 167–170.
Akyildiz, I.F., Lee, A., Wang, P., Luo, M., Chou, W., 2016. Research challenges for traffic engineering in software defined networks. IEEE Netw. 30 (3), 52–58.
Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A., 2010. Hedera: Dynamic flow scheduling for data center networks. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, pp. 19–19.
Arajo, J.T., Landa, R., Clegg, R.G., Pavlou, G., 2014. Software-defined network support for transport resilience. In: Proceedings of Network Operations and Management Symposium, pp. 1–8.
Aslan, M., Matrawy, A, 2017. A clustering-based consistency adaptation strategy for distributed SDN controllers, http://arXiv:1705.09050.
Bannour, F., Souihi, S., Mellouk, A., 2017. Software-defined networking: A self-adaptive consistency model for distributed SDN controllers, RESCOM 2017 201 16.
Bari, M.F., Roy, A.R., Chowdhury, S.R., Zhang, Q., 2013. Dynamic controller provisioning in software defined networks. In: Proceedings of IEEE/ACM/IFIP International Conference on Network and Service Management, pp. 18–25.
Beheshti, N., Zhang, Y., 2012. Fast failover for control traffic in software-defined networks. In: Global Communications Conference, pp. 2665–2670.
Benamrane, F., Benaini, R., et al., 2017. An east-west interface for distributed SDN control plane: implementation and evaluation. Comput. Electr. Eng. 57, 162–175.
Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., 2014. Onos: towards an open, distributed SDN os. In: Proceedings of the Workshop on Hot Topics in Software Defined Networking, pp. 1–6.
Berman, M., Chase, J.S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., Seskar, I., 2014. Geni: a federated testbed for innovative network experiments. Comput. Netw. 61, 5–23.
Borokhovich, M., Schiff, L., Schmid, S., 2014. Provable data plane connectivity with local fast failover: Introducing OpenFlow graph algorithms. In: Proceedings of the Third Workshop on Hot topics in Software Defined Networking, ACM, pp. 121–126.
Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al., 2014. P4: programming protocol-independent packet processors. ACM SIGCOMM Comput. Commun. Rev. 44 (3), 87–95.
Botelho, F., Bessani, A., Ramos, F., Ferreira, P., 2014. Smartlight: A practical fault-tolerant sdn controller, In: Proceedings 3rd European Workshop Software Defined Networks, p.6.
Botelho, F., Valente Ramos, F.M., Kreutz, D., Bessani, A., 2013. On the feasibility of a consistent and fault-tolerant data store for SDNs. In: Proceedings of Second European Workshop on Software Defined Networks, pp. 38–43.
Bozakov, Z., Papadimitriou, P., 2012. Autoslice: automated and scalable slicing for software-defined networks. In: Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop, ACM, pp. 3–4.
Canini, M., Kuznetsov, P., Levin, D., Schmid, S., 2015. A distributed and robust SDN control plane for transactional network updates. In: Proceedings of IEEE Conference on Computer Communications (INFOCOM), IEEE, pp. 190–198.
Cello, M., Xu, Y., Walid, A., Wilfong, G., Chao, H.J., Marchese, M., 2017. Balcon: A distributed elastic SDN control via efficient switch migration. In: Proceedings of IEEE International Conference on Cloud Engineering (IC2E), 2017 , pp. 40–50.
Chang, Y., Rezaei, A., Vamanan, B., Hasan, J., Rao, S., Vijaykumar, T., 1609. Hydra: Leveraging functional slicing for efficient distributed SDN controllers, http://arXiv:1609.07192, 2016.
Chauhan, O., Mamoun, M.B., Benaini, R., 2016. An overview on SDN architectures with multiple controllers. J. Comput. Netw. Commun. 2016 2, 1–8.
Chen, J., Chen, J., Ling, J., Zhang, W., 2016. Failure recovery using vlan-tag in SDN: High speed with low memory requirement. In: Proceedings of IEEE 35th International Performance Computing and Communications Conference (IPCCC), pp. 1–9.
Cheng, G., Chen, H., Wang, Z., Chen, S., 2015. Dha: Distributed decisions on the switch migration toward a scalable SDN control plane. In: Proceedings of IFIP Networking Conference (IFIP Networking), IEEE, pp. 1–9.
Chowdhury, N.M.M.K., Boutaba, R., 2010. A survey of network virtualization. Comput. Netw. 54 (5), 862–876.
Controller platform (oscp): Clustering, ⟨http://https://wiki.opendaylight.org/view/OpenDaylight_Controller/Clustering⟩, (accessed 17 April 2001) (2013).
Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S., 2011. Devoflow: Scaling flow management for high-performance networks. In: ACM SIGCOMM Computer Communication Review, Vol. 41, ACM, pp. 254–265.
Dacier, M.C., König, H., Cwalinski, R., Kargl, F., Dietrich, S., 2017. Security challenges and opportunities of software-defined networking. IEEE Secur. Priv. 15 (2), 96–100.
Dan, L., Wundsam, A., Heller, B., Handigol, N., 2012. A. Feldmann, Logically centralized? State distribution trade-offs in software defined networks, pp. 1–6.
Davie, B., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Gude, N., Padmanabhan, A., Petty, T., Duda, K., Chanda, A., 2017. A database approach to SDN control plane design. ACM SIGCOMM Comput. Commun. Rev. 47 (1), 15–26.
Dixit, A., Hao, F., Mukherjee, S., Lakshman, T.V., Kompella, R., 2013. Towards an elastic distributed SDN controller. Acm Sigcomm Comput. Commun. Rev. 43 (4), 7–12.
Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., Kompella, R.R., 2014. Elasticon; an elastic distributed SDN controller. In: Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 17–27.
Duan, Q., Ansari, N., Toy, M., 2016. Software-defined network virtualization: an architectural framework for integrating sdn and nfv for service provisioning in future networks. IEEE Netw. 30 (5), 10–16.
Farhady, H., Lee, H.Y., Nakao, A., 2015. Software-defined networking: a survey. Comput. Netw. Int. J. Comput. Telecommun. Netw. 81 (C), 79–95.
Ferguson, A.D., Guha, A., Liang, C., Fonseca, R., Krishnamurthi, S., 2013. Participatory networking: An api for application control of SDNs. In: ACM SIGCOMM Computer

Communication Review, Vol. 43, ACM, pp. 327–338.
FlowVisor source code, 2012. ⟨https://github.com/opennetworkinglab/flowvisor⟩.
Fonseca, P., Bennesby, R., Mota, E., Passito, A., 2012. A replication component for resilient OpenFlow-based networking. In: Proceedings of Network Operations and Management Symposium, pp. 933–939.
Fonseca, P., Bennesby, R., Mota, E., Passito, A., 2013. Resilience of SDNs based on active and passive replication mechanisms. In: Proceedings of Global Communications Conference, pp. 2188–2193.
Förster, K.-T., Mahajan, R., Wattenhofer, R., 2016. Consistent updates in software defined networks: on dependencies, loop freedom, and blackholes. In: Proceedings of IFIP Networking Conference (IFIP Networking) and Workshops, IEEE, pp. 1–9.
Fratczak, T., Broadbent, M., Georgopoulos, P., Race, N., 2013. Homevisor: Adapting home network environments. In: Proceedings of the Second European Workshop on Software Defined Networks (EWSDN), IEEE, pp. 32–37.
Gray, N., Zinner, T., Gebert, S., Tran-Gia, P., 2016. Simulation framework for distributed SDN-controller architectures in omnet++. In: International Conference on Mobile Networks and Management, Springer, pp. 3–18.
Guan, J., Ma, S., Guo, B., Li, J., Huang, S., Cao, X., Chen, Z., Li, Z., He, Y., 2013. First field demonstration of network function virtualization via dynamic optical networks with opencontrail and enhanced nox orchestration. In: Proceedings of Asia Communications and Photonics Conference, Optical Society of America, pp. AF2C-5.
Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Mckeown, N., Shenker, S., 2008. Nox: towards an operating system for networks. ACM Sigcomm Comput. Commun. Rev. 38 (3), 105–110.
Guha, A., Reitblatt, M., Foster, N., 2013. Machine-verified network controllers. In: ACM SIGPLAN Notices, Vol. 48, ACM, pp. 483–494.
Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D.C., Gayraud, T., 2014. Software-defined networking: challenges and research opportunities for future internet. Comput. Netw. 75, 453–471.
Hark, R., Stingl, D., Richerzhagen, N., Nahrstedt, K., Steinmetz, R., 2016. DISTTM: Collaborative traffic matrix estimation in distributed SDN control planes. In: Proceedings of IFIP Networking Conference (IFIP Networking) and Workshops, 2016, IEEE, pp. 82–90.
Hassas Yeganeh, S., Ganjali, Y., 2012. Kandoo: a framework for efficient and scalable offloading of control applications. In: Proceedings of the Workshop on Hot Topics in Software Defined Networks, pp. 19–24.
He, M., Basta, A., Blenk, A., Kellerer, W., 2017. Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows. In: Proceedings of IEEE International Conference on Communications (ICC), pp. 1–7.
Heller, B., Sherwood, R., Mckeown, N., 2012. The controller placement problem. ACM SIGCOMM Comput. Commun. Rev. 42 (4), 473–478.
Ho, C.-C., Wang, K., Hsu, Y.-H., 2016. A fast consensus algorithm for multiple controllers in software-defined networks. In: Proceedings of the 18th International Conference on Advanced Communication Technology (ICACT), IEEE, pp. 112–116.
Hock, D., Gebert, S., Hartmann, M., Zinner, T., Tran-Gia, P., 2014. Poco-framework for pareto-optimal resilient controller placement in SDN-based core networks 1–2.
Hock, D., Hartmann, M., Gebert, S., Jarschel, M., Zinner, T., Tran-Gia, P., 2013. Pareto-optimal resilient controller placement in SDN-based core networks. In: Proceedings of 25th International Teletraffic Congress (ITC), IEEE, pp. 1–9.
Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., Wattenhofer, R., 2013. Achieving high utilization with software-driven wan. In: ACM SIGCOMM Computer Communication Review, Vol. 43, ACM, 2013, pp. 15–26.
Hu, J., Lin, C., Li, X., Huang, J., 2014. Scalability of control planes for software defined networks: modeling and evaluation. Qual. Serv., 147–152.
Hu, Y., Wang, W., Gong, X., Que, X., Cheng, S., 2014. On reliability-optimized controller placement for software-defined networks. China Commun. 11 (2), 38–54.
Hu, Y., Wang, W., Gong, X., Que, X., 2013. Reliability-aware controller placement for software-defined networks. In: Proceedings of IFIP/IEEE International Symposium on Integrated Network Management, pp. 672–675.
Hu, Y., Wang, W., Gong, X., Que, X., Cheng, S., 2012. Balanceflow: Controller load balancing for OpenFlow networks. In: Proceedings of IEEE International Conference on Cloud Computing and Intelligent Systems, pp. 780–785.
HyperFlow source code, 2013. ⟨https://github.com/dice-cyfronet/hyperflow⟩.
IRIS source code, 2013. ⟨http://openiris.etri.re.kr/,ProjectOpenIRIS⟩.
Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al., 2013. B4: experience with a globally-deployed software defined wan, ACM SIGCOMM. Comput. Commun. Rev. 43 (4), 3–14.
Jiang, G.L., Wu, J., Liu, K., Fu, B.Z., Chen, M.Y., 2014. Implementation and performance analysis of SDN controller. Dongbei Daxue Xuebao/J. Northeast. Univ. 35, 313–318.
Jimenez, Y., Cervello-Pastor, C., Garcia, A.J., 2014. On the controller placement for designing a distributed SDN control layer. In: Proceedings of Networking Conference, 2014 IFIP, IEEE, pp. 1–9.
Johns, M., 2013. Getting Started with Hazelcast, Packt Publishing Ltd.
Kandoo source code, ⟨https://github.com/kandoo⟩, 2015.
Karakus, M., Durresi, A., 2017. Quality of service (qos) in software defined networking (SDN): a survey. J. Netw. Comput. Appl. 80, 200–218.
Kaur, S., Singh, J., Ghumman, N.S., 2014. Network programmability using pox controller. In: Proceedings of ICCCS International Conference on Communication, Computing & Systems, IEEE, no. s 134, p. 138.
Kazemian, P., Varghese, G., McKeown, N., 2012. Header space analysis: Static checking for networks.. In: Proceedings of NSDI, vol. 12, pp. 113–126.
Kempf, J., Bellagamba, E., Kern, A., Jocha, D., 2012. Scalable fault management for OpenFlow. In: Proceedings of IEEE International Conference on Communications, pp. 6606–6610.
Khurshid, A., Zhou, W., Caesar, M., Godfrey, P., 2012. Veriflow: verifying network-wide invariants in real time. ACM SIGCOMM Comput. Commun. Rev. 42 (4), 467–472.
Kim, H., Feamster, N., 2013. Improving network management with software defined networking. Commun. Mag. IEEE 51 (2), 114–119.
Kim, T., Choi, S.-G., Myung, J., Lim, C.-G., 2017. Load balancing on distributed datastore in opendaylight SDN controller cluster. In: Proceedings of IEEE Conference on Network Softwarization (NetSoft), pp. 1–3.
Koohanestani, A.K., Osgouei, A.G., Saidi, H., Fanian, A., 2017. An analytical model for delay bound of OpenFlow based SDN using network calculus. J. Netw. Comput. Appl. 96, 31–38.
Koponen, T., Amidon, K., Balland, P., Casado, M., Chanda, A., Fulton, B., Ganichev, I., Gross, J., Gude, N., Ingram, P., et al., 2014. Network virtualization in multi-tenant datacenters. In: Proceedings of USENIX NSDI.
Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al., 2010. Onix: A distributed control platform for large-scale production networks. In: Proceedings of OSDI, vol. 10, pp. 1–6.
Koshibe, A., Baid, A., Seskar, I., 2014. Towards distributed hierarchical SDN control plane. In: Proceedings of First International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), IEEE, pp. 1–5.
Kreutz, D., Ramos, F.M.V., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S., 2014. Software-defined networking: a comprehensive survey. Proc. IEEE 103 (1), 10–13.
Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S., 2015. Software-defined networking: a comprehensive survey. Proc. IEEE 103 (1), 14–76.
Krishnamurthy, A., Chandrabose, S.P., Gember-Jacobson, A., 2014. Pratyaastha: An efficient elastic distributed SDN control plane. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ACM, pp. 133–138.
Kyung, Y., Hong, K., Nguyen, T.M., Park, S., 2015. A load distribution scheme over multiple controllers for scalable SDN. In: Proceedings of the Seventh International Conference on Ubiquitous and Future Networks, pp. 808–810.
Lakshman, A., Malik, P., 2010. Cassandra: a decentralized structured storage system. ACM SIGOPS Oper. Syst. Rev. 44 (2), 35–40.
Lam, J.-H., Lee, S.-G., Lee, H.-J., Oktian, Y.E., 2015. Securing distributed SDN with ibc. In: Proceedings of the Seventh International Conference on Ubiquitous and Future Networks (ICUFN), IEEE, pp. 921–925.
Lam, J.H., Lee, S.-G., Lee, H.-J., Oktian, Y.E., 2016. Tls channel implementation for onoss east/west-bound communication. In: Proceedings of Electronics, Communications and Networks V, Springer, pp. 397–403.
Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., 2015. Heuristic approaches to the controller placement problem in large scale SDN networks. IEEE Trans. Netw. Serv. Manag. 12 (1), 4–17.
Lange, S., Gebert, S., Spoerhase, J., Rygielski, P., Zinner, T., Kounev, S., Tran-Gia, P., 2015. Specialized heuristics for the controller placement problem in large scale SDN networks. In: 27th International Teletraffic Congress (ITC 27), IEEE, pp. 210–218.
Lee, B., Park, S.H., Shin, J., Yang, S., 2014. Iris: the OpenFlow-based recursive SDN controller. In: Proceedings of the 16th International Conference on Advanced Communication Technology (ICACT), IEEE, pp. 1227–1231.
Li, W., Meng, W., Kwok, L.F., 2016. A survey on OpenFlow-based software defined networks: security challenges and countermeasures. J. Netw. Comput. Appl. 68, 126–139.
Li, H., De Grande, R.E., Boukerche, A., 2017. An efficient cpp solution for resilience-oriented SDN controller deployment. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 540–549.
Liao, J., Sun, H., Wang, J., Qi, Q., Li, K., Li, T., 2017. Density cluster based approach for controller placement problem in large-scale software defined networkings. Comput. Netw. 112, 24–35.
Lin, P., Bi, J., Wolff, S., Wang, Y., 2015. A west-east bridge based SDN inter-domain testbed. IEEE Commun. Mag. 53 (2), 190–197.
Liu, W., Bobba, R.B., Mohan, S., Campbell, R.H., 2015. Inter-flow consistency: A novel SDN update abstraction for supporting inter-flow constraints. In: Proceedings of IEEE Conference on Communications and Network Security (CNS), pp. 469–478.
Lopes, F.A., Santos, M., Fidalgo, R., Fernandes, S., 2016. A software engineering perspective on SDN programmability. IEEE Commun. Surv. Tutor. 18 (2), 1255–1272.
Masoudi, R., Ghaffari, A., 2016. Software defined networks: a survey. J. Netw. Comput. Appl. 67, 1–25.
Matsumoto, S., Hitz, S., Perrig, A., 2014. Fleet: Defending SDNs from malicious administrators. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ACM, pp. 103–108.
Mcgeer, R., 2012. A safe, efficient update protocol for OpenFlow networks. In: Proceedings of the Workshop on Hot Topics in Software Defined Networks, pp. 61–66.
McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J., 2008. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. 38 (2), 69–74.
Medved, J., Varga, R., Tkacik, A., Gray, K., 2014. Opendaylight: towards a model-driven SDN controller architecture. World Wirel., Mob. Multimed. Netw., 1–6.
Morales, L.V., Murillo, A.F., Rueda, S.J., 2015. Extending the floodlight controller. In: Proceedings of IEEE International Symposium on Network Computing and Applications, pp. 126–133.
Muqaddas, A.S., Giaccone, P., Bianco, A., Maier, G., 2017. Inter-controller traffic to support consistency in onos clusters. IEEE Trans. Netw. Serv. Manag. PP, 1.
Nencioni, G., Helvik, B.E., Gonzalez, A.J., Heegaard, P.E., Kamisinski, A., 2017. Impact of sdn controllers deployment on network availability, arXiv:1703.05595.
Nunes, A., Mendonca, M., Nguyen, X.N., Obraczka, K., 2014. A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun. Surv. Tutor. 16 (3), 1617–1634.
of13softswitch, ⟨https://github.com/CPqD/ofsoftswitch13⟩, (accessed 7 September

2017) (2013).
Oktian, Y.E., Lee, S., Lee, H., Lam, J., 2017. Distributed SDN controller system: a survey on design choice. J. Lam, Distrib. 121, 100–111.
ONOS Distributed Primitives, 2015. ⟨https://wiki.onosproject.org/display/ONOS/Distributed+Primitives⟩, (accessed 22 September 2017).
ONOS source code, 2014. ⟨https://wiki.onosproject.org/display/ONOS/Downloads⟩.
Open Networking Foundation: ONF, 2011. ⟨https://www.opennetworking.org⟩, (accessed 17 March 2016).
OpenDayligh source code, 2013. ⟨https://www.opendaylight.org/technical-community/getting-started-for-developers/downloads-and-documentation⟩.
OpenFlow Specification 1.0., 2009. ⟨http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf⟩., (accessed 5 April 2017).
OpenFlow Specification 1.1.0., 2011. ⟨http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf⟩., (accessed 5 April 2017) .
OpenFlow Specification 1.2.0., 2011. ⟨https://www.opennetworking.org/images/stories/downloads/specification/openflow-specv1.2.pdf⟩., (accessed 5 April 2017).
OpenFlow Specification 1.3.0., 2012. ⟨https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf⟩., (accessed 5 April 2017).
OpenFlow Specification 1.4.0., 2013. ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf⟩, (accessed 7 April 2017).
OpenFlow Specification 1.5.0., 2014. ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf⟩, (accessed 7 April 2017).
OpenFlow-Configuration Protocol 1.2, 2014. ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf⟩, (accessed 7 April 2017).
Panda, A., Zheng, W., Hu, X., Krishnamurthy, A., Shenker, S., 2017. Scl: Simplifying distributed SDN control planes. In: Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), USENIX Association, pp. 329–345.
Perešíni, P., Kuzniar, M., Vasić, N., Canini, M., Kostiū, D., 2013. of. cpp: Consistent packet processing for OpenFlow. In: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, pp. 97–102.
Phemius, K., Bouet, M., Leguay, J., 2013. Disco: Distributed multi-domain SDN controllers. In: Proceedings of the Network Operations and Management Symposium, pp. 1–4.
Qi, C., Wu, J., Hu, H., Cheng, G., Liu, W., Ai, J., Yang, C., 2016. An intensive security architecture with multi-controller for sdn. In: Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 401–402.
Raeisi, B., Giorgetti, A., 2016. Software-based fast failure recovery in load balanced sdn-based datacenter networks. In: Proceedings of the International Conference on Information Communication and Management (ICICM), IEEE, pp. 95–99.
Ramos, R.M., Martinello, M., Esteve Rothenberg, C., 2013. Slickflow: Resilient source routing in data center networks unlocked by OpenFlow. In: Local Computer Networks, pp. 606–613.
Rath, H.K., Revoori, V., Nadaf, S.M., Simha, A., 2014. Optimal controller placement in software defined networks (SDN) using a non-zero-sum game. In: Proceedings of IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks, 2014, pp. 1–6.
Reitblatt, M., Foster, N., Rexford, J., Schlesinger, C., Walker, D., 2012. Abstractions for network update. ACM Sigcomm Comput. Commun. Rev. 42 (4), 323–334.
Reitblatt, M., Foster, N., Rexford, J., Walker, D., 2011. Consistent updates for software-defined networks: change you can believe in!. In: Proceedings of ACM Workshop on Hot Topics in Networks, p. 7.
Ros, F.J., Ruiz, P.M., 2014. Five nines of southbound reliability in software-defined networks. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ACM, pp. 31–36.
Sakic, E., Sardis, F., Guck, J.W., Kellerer, W., 2017. Towards adaptive state consistency in distributed SDN control plane. In: Proceedings of IEEE International Conference on Communications (ICC), pp. 1–7.
Sallahi, A., Sthilaire, M., 2015. Optimal model for the controller placement problem in software defined networks. In: Proceedings of ACM International Conference on Multimodal Interaction, pp. 30-33.
Schiff, L., Schmid, S., Canini, M., 2016. Ground control to major faults: Towards a fault tolerant and adaptive sdn control network. In: Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, pp. 90–96.
Schmid, S., Suomela, J., 2013. Exploiting locality in distributed SDN control. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ACM, pp. 121–126.
Scott-Hayward, S., 2015. Design and deployment of secure, robust, and resilient SDN controllers. In: Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1–5.
SDN architecture Overview, 2014. ⟨https://www.opennetworking.org/wp-content/uploads/2013/02/SDN-architecture-overview-1.0.pdf⟩, (accessed 7 September 2017).
Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., 2013. OpenFlow-based segment protection in ethernet networks. IEEE/OSA J. Opt. Commun. Netw. 5 (9), 1066–1075.
Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., Smeliansky, R., 2013. Advanced study of SDN/OpenFlow controllers. In: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, ACM, p. 1.
Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P., 2013. OpenFlow: meeting carrier-grade recovery requirements. Comput. Commun. 36 (6), 656–665.
Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G., 2009. Flowvisor: a network virtualization layer. Open. Switch Consort., Tech. Rep., 1–13.
Song, P., Liu, Y., Liu, T., Qian, D., 2017a. Flow stealer: lightweight load balancing by stealing flows in distributed SDN controllers. Sci. China Inform. Sci. 60 (3), 032202.
Song, P., Liu, Y., Liu, C., Qian, D., 2017b. Paraflow: fine-grained parallel SDN controller for large-scale networks. J. Netw. Comput. Appl. 87, 46–59.
Spalla, E.S., Mafioletti, D.R., Liberato, A.B., Rothenberg, C., Camargos, L., Villaca, R.D. S., Martinello, M., 2015. Resilient strategies to SDN: An approach focused on actively replicated controllers. In: Proceedings of Brazilian Symposium on Computer Networks & Distributed Systems, pp. 246–259.
Sridharan, V., Gurusamy, M., Truong-Huu, T., 2017. On multiple controller mapping in software defined networks with resilience constraints. IEEE Commun. Lett. 21, 1763–1766.
Thakur, A., Goraya, M.S., 2017. A taxonomic survey on load balancing in cloud. J. Netw. Comput. Appl. 98, 43–57.
Tootoonchian, A., Ganjali, Y., 2010. Hyperflow: A distributed control plane for OpenFlow. In: Proceedings of the Internet Network Management Conference on Research on Enterprise Networking, pp. 3–3.
Toumi, K., Idrees, M.S., Charmet, F., Yaich, R., Blanc, G., 2016. Usage control policy enforcement in sdn-based clouds: A dynamic availability service use case. In: High Performance Computing and Communications; In: Proceedings of the 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on, IEEE, pp. 578–585.
Tuncer, D., Charalambides, M., Clayman, S., Pavlou, G., 2015. On the placement of management and control functionality in software defined networks. In: International Conference on Network and Service Management, pp. 360–365.
Vanbever, L., Reich, J., Benson, T., Foster, N., Rexford, J., 2013. Hotswap: Correct and efficient controller upgrades for software-defined networks. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ACM, pp. 133–138.
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.
Wang, J., Shou, G., Hu, Y., Guo, Z., 2016. A multi-domain sdn scalability architecture implementation based on the coordinate controller. In: Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), IEEE, pp. 494–499.
Wang, T., Liu, F., Guo, J., Xu, H., 2016. Dynamic SDN controller assignment in data center networks: Stable matching with transfers. In: Proceedings of the 35th Annual IEEE International Conference on Computer Communications, IEEE INFOCOM, pp. 1–9.
Wang, Y., Matta, I., 2014. SDN management layer: Design requirements and future direction. In: IEEE International Conference on Network Protocols, pp. 555–562.
Wibowo, F.X., Gregory, M.A., Ahmed, K., Gomez, K.M., 2017. Multi-domain software defined networking: research status and challenges. J. Netw. Comput. Appl. 87, 32–45.
Williams, D., Jamjoom, H., 2013. Cementing high availability in OpenFlow with rulebricks. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ACM, pp. 139–144.
Wundsam, A., Levin, D., Seetharaman, S., Feldmann, A., et al., 2011. Ofrewind: Enabling record and replay troubleshooting for networks. In: Proceedings of USENIX Annual Technical Conference, pp. 15–17.
Xia, W., Wen, Y., Foh, C.H., Niyato, D., 2015. A survey on software-defined networking. IEEE Commun. Surv. Tutor. 17 (1), 27–51.
Xiao, P., Qu, W., Qi, H., Li, Z., 2014. The SDN controller placement problem for wan. In: Proceedings IEEE/CIC International Conference on Communications in China, pp. 220–224.
Xu, G., Dai, B., Huang, B., Yang, J., Wen, S., 2017. Bandwidth-aware energy efficient flow scheduling with SDN in data center networks. Future Gener. Comput. Syst. 68, 163–174.
Yan, Q., Yu, F.R., Gong, Q., Li, J., 2016. Software-defined networking (SDN) and distributed denial of service (ddos) attacks in cloud computing environments: a survey, some research issues, and challenges. IEEE Commun. Surv. Tutor. 18 (1), 602–622.
Yao, G., Bi, J., Li, Y., Guo, L., 2014. On the capacitated controller placement problem in software defined networks. Commun. Lett. IEEE 18 (8), 1339–1342.
Yao, G., Bi, J., Guo, L., 2013. On the cascading failures of multi-controllers in software defined networks. In: Proceedings of IEEE International Conference on Network Protocols, pp. 1–2.
Yao, L., Hong, P., Zhang, W., Li, J., Ni, D., 2015. Controller placement and flow based dynamic management problem towards SDN. In: IEEE International Conference on Communication Workshop (ICCW), pp. 363–368.
Ye, X., Cheng, G., Luo, X., Maximizing, 2017. SDN control resource utilization via switch migration. Comput. Netw. 126, 69–80.
Yeganeh, S.H., Tootoonchian, A., Ganjali, Y., 2013. On scalability of software-defined networking. IEEE Commun. Mag. 51 (2), 136–141.
Yin, H., Xie, H., Tsou, T., Lopez, D., Aranda, P., Sidi, R., 2012. SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. 2015, from ⟨https://datatracker.ietf.org/doc/draft-yin-sdn-sdni/⟩.
Yu, M., Rexford, J., Freedman, M.J., Wang, J., 2011. Scalable flow-based networking with difane. ACM SIGCOMM Comput. Commun. Rev. 41 (4), 351–362.
Yu, H., Li, K., Qi, H., Li, W., Tao, X., 2015. Zebra: an east-west control framework for sdn controllers. In: Proceedings of the 44th International Conference on Parallel Processing (ICPP), pp. 610–618.
Yu, J., Wang, Y., Pei, K., Zhang, S., Li, J., 2016. A load balancing mechanism for multiple SDN controllers based on load informing strategy. In: Proceedings of 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), IEEE, pp. 1–4.

Zhang, T., Giaccone, P., Bianco, A., De Domenico, S., 2017. The role of the inter-controller consensus in the placement of distributed SDN controllers. Comput. Commun. 113, 1–13.
Zhang, S., Wang, Y., He, Q., Yu, J., Guo, S., 2016. Backup-resource based failure recovery approach in sdn data plane. In: Proceedings of 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), IEEE, pp. 1–6.
Zhang, Y., Beheshti, N., Tatipamula, M., 2011. On resilience of split-architecture networks. In: Proceedings of Global Telecommunications Conference (GLOBECOM 2011), IEEE, pp. 1–6.
Zhao, Z., Wu, B., 2017. Scalable SDN architecture with distributed placement of controllers for wan, Concurrency and Computation: Practice and Experience.
Zhong, H., Sheng, J., Cui, J., Xu, Y., 2017. Scplbs: A smart cooperative platform for load balancing and security on SDN distributed controllers. In: Proceedings of the 3rd IEEE International Conference on Cybernetics (CYBCONF), IEEE, pp. 1–6.
Zhou, Y., Zhu, M., Xiao, L., Li, R., Duan, W., Li, D., Liu, R., Zhu, M., 2014. A load balancing strategy of SDN controller based on distributed decision. In: Proceedings of IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 851–856.

Zhang Yuan, born in 1994. She is currently a master candidate in Jinan University. Her research interests include software defined networking (SDN) and data center networking.

Cui Lin, born in 1985. He is currently an Associate Professor in Jinan University. He obtained Ph.D. degree in City University of Hong Kong in 2013. His main research interests include cloud data center resource management, data center networking, software defined networking (SDN), virtualization, distributed systems as well as wireless networking.

Wang Wei, born in 1982. He is currently a network engineer in Network and Information Management Center of Shandong University at Wei Hai. He received both master and bachelor degree from Shandong University in 2004 and 2010 respectively. His main research interests include networking resource schedule in enterprise networks.

Yuxiang Zhang, born in 1992. He is currently pursuing the Master Degree in computer science in Jinan University. He received the B.E. degree in network engineering from Jinan University, China, in 2013. His current research interests are in the area of data center networks.