Journal of Network and Computer Applications 141 (2019) 23–58

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Challenges and solutions in Software Defined Networking: A survey

Surbhi Saraswat, Vishal Agarwal, Hari Prabhat Gupta, Rahul Mishra *, Ashish Gupta, Tanima Dutta

Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, India

ARTICLE INFO

Keywords:
Design issues
Research challenges
Software defined network
Solutions

ABSTRACT

Software Defined Network (SDN) has become one of the most preferred solutions for the management of large-scale complex networks. The network policies in the large-scale network are difficult to embed on entire network devices simultaneously, whereas in SDN these policies can be embedded on the top of the network. The SDN network is divided into two parts which are vertically integrated to form the entire network. The policies and control mechanism are handled by the top layer called the control plane, whereas the data forwarding is performed at the bottom layer called the data plane. Thus the SDN forms a logically centralized network. In this paper, we present a comprehensive survey on the various issues, challenges, and solutions in the designing, implementation, performance, and verification of SDN. The network design emphasized on scalability, fault-tolerance, flexibility, and elasticity. Further implementation issues discuss resource management and virtualization. Similarly, latency, efficiency, and consistency discuss under-performance measure. Lastly, we focus on security and debugging in verification of SDN. We elaborate all the above issues and further discuss the different solutions that exist and discussed the future research trends in SDN.

1. Introduction

The introduction of Software Defined Networking (SDN) commences a paradigm shift in communication networks (Xia et al., 2015; Lopes et al., 2016; Li et al., 2014a; Kreutz et al., 2015). This shift introduces an abstraction towards a more centralized approach of network control, where a controller manages and operates a network through open interfaces. The SDN architecture comprises of application, control, and infrastructure layers as shown in Fig. 1. The logically centralized architecture separates the control and infrastructure layers by using Southbound Application Programming Interfaces (API). Such separation supports programmability of the control plane that allows moving control plane functions from network devices to dedicated controller instances running in software. The key difference between traditional network and SDN is: *how to handle the data packets?*. In a traditional network, the way a switch handles an incoming data packet is written into its firmware. The SDN forwards data packets to the controller for further processing.

The SDN provides the granular control of the controller over the way in which the switches handle the packets and the ability to automatically prioritize or block some packets. It provides greater efficiency without the need to invest in expensive and application-specific network switches. The SDN encompasses multiple types of network technologies designed to make the network more agile. It is also flexible to support the virtualization of server and storage infrastructure of the data center. The SDN is defined as an approach for designing, building, and managing networks that separates the control of network and forwarding planes. It enables the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.

Several benefits that can be achieved by implementing SDN in place of traditional network in a multi-tenant network environment, are as follows:

- *Granularity*: SDN allows policies to be applied at granule levels (*e.g.,* session, user, device, application) in a highly abstracted and automated manner.
- *Utilization*: SDN switches in the network can change their functionality based on the instructions given by the controller. This will increase the utilization of available resources inside the SDN.
- *Complexity*: SDN reduces the complexity of network by decreasing operational overhead and network instability (introduced by het-

* Corresponding author.
E-mail addresses: surbhis.rs.cse16@iitbhu.ac.in (S. Saraswat), vishal.agarwal.cse13@iitbhu.ac.in (V. Agarwal), hariprabhat.cse@iitbhu.ac.in (H.P. Gupta), rahulmishra.rs.cse17@iitbhu.ac.in (R. Mishra), ashishg.rs.cse16@iitbhu.ac.in (A. Gupta), tanima.cse@iitbhu.ac.in (T. Dutta).

https://doi.org/10.1016/j.jnca.2019.04.020
Received 10 October 2018; Received in revised form 24 March 2019; Accepted 23 April 2019
Available online 12 May 2019

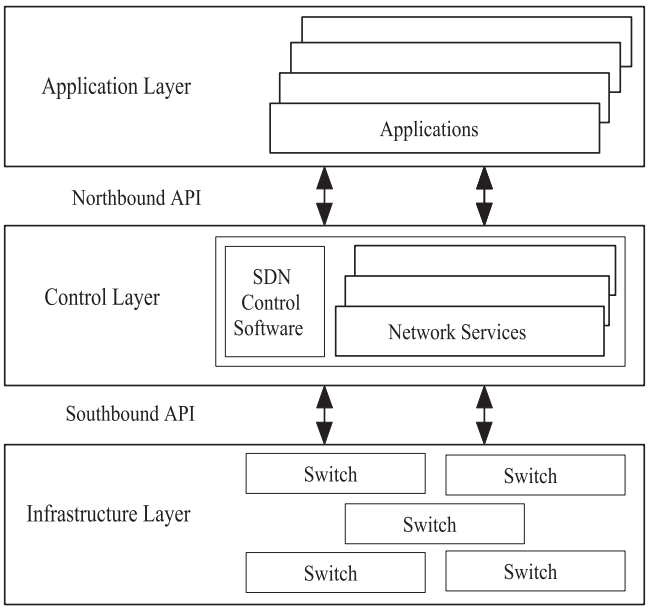


Fig. 1. Illustration of a three-layer SDN architecture.

erogeneous devices). The centralized controller in SDN reduces the complexity of configuring entire network devices for a small change in the network policy (*e.g.,* scheduling policy for some important task).

- *Centralized network provisioning*: SDN makes a centralized view of entire network and thus helpful in centralized enterprise management and provisioning. Inspite of managing group of devices by a single vendor, SDN can be used to manage different vendor devices.
- *Cost savings*: SDN leads to save overall cost of network establishment. It reduces the spending required on infrastructure by allowing data centres to get the most use of their existing devices.

Network designing, implementation, performance monitoring, and verification are the main phases required to setup the SDN. A network designer collects the requirements of the users and designs a network that can handle upcoming challenges of the network. To meet the centralized management, the network designing of SDN must satisfy the desired Quality of Services (QoS) and therefore it becomes quite complex. A network implementation is a step which converses the network designed plan to real world solution. Network performance phase refers to measures of QoS of the network as seen by the users. In the network performance phase latency, efficiency, and consistency are the main performance metrics used for measuring the quality of the network. Network verification phase is used to identify whether the designed network is suitable for providing the desired QoS. Compare with traditional network, the SDN verification problem is complex because network states in SDN are dynamic. The SDN generates faults (*e.g.,* conflicting rules, inconsistent network view) because it allows multiple applications to simultaneously program the data plane. Testing, debugging, and security are the crucial features for network verification, as it justifies the reliability of the network.

This paper presents the existing challenges and solutions to setup of a new SDN that satisfy the desired QoS. The next section discusses the network designing, implementation, performance monitoring, and verification phases in details. The network designing covers the scalability, flexibility, fault tolerance, and consistency issues of the network. We discuss the network designing challenges and solutions in Section 3. While implementing the SDN in real world, the challenges faced by network designers include integration of SDN with traditional or existing network, managing resources among devices, and managing virtual networks. Section 4 covers such challenges that are faced in implementation the SDN. The challenges for measuring the performance of the SDN has covered in 5. The crucial features of network verification are discussed in Section 6. Finally, the conclusion of the paper is given in Section 7.

2. SDN challenges and research directions

The SDN improves the performance of the network, provides better network control, and an area for innovation. Despite the benefits, it is still in early stage of development and various issues and challenges are yet to overcome. Table 1 categorizes the challenges faced in the growth and development of SDN. The following subsections briefly discuss such issues and challenges in designing of SDN.

2.1. Network design

SDN provides a centralized view of the entire network, which makes it easier to single point control. SDN must be designed and operated to meet the growing requirements of the users using the available resources. Examples of such requirements are around-the-clock service and dedicated link between controller and switches. It shows that the controller must be available all the time for a given user and the switches. The requirements of the users also raise the challenges in the network, *e.g.*, managing large network size, reliability, robustness, and ability of the real time changes in the network. Due to the large size networks, it is no longer practical to construct networks by connecting many standalone components without careful planning and design. The requirements of the users can be translated into the following fundamental network design goals:

- **Scalability:** Scalable network designs can grow to include the network load and support new applications without impacting the level of service delivered to existing users (Bondi, 2000). A system with high scalability maintains its efficiency even when the number

Table 1
Illustration of the existing work classified according to available challenges in SDN.

SDN Challenges		
	Network Design (Section 3)	Scalability (Section 3.1)
		Fault-Tolerance (Section 3.2)
		Flexibility (Section 3.3)
		Elasticity (Section 3.4)
	Network Implementation (Section 4)	Integration With traditional networks (Section 4.1)
		Resource Management (Section 4.2)
		Virtualization (Section 4.3)
	Network Performance (Section 5)	Latency (Section 5.1)
		Efficiency (Section 5.2)
		Consistency (Section 5.3)
		Traffic Measurements (Section 5.4)
	Network Verification (Section 6)	Hardware Testing (Section 6.1)
		Debugging (Section 6.2)
		Security (Section 6.3)

Table 2
Network design challenges in SDN.

Network Design	Scalability	Centralized Control SDN (Bifulco et al., 2016; Curtis et al., 2011)
		Flat Model Distribution Controllers SDN (Dixit et al., 2014a; Tootoonchian and Ganjali, 2010; Koponen et al., 2010; Yazici et al., 2014; Tam et al., 2011; Phemius et al., 2014)
		Hierarchical Distributed Controllers SDN (Hassas Yeganeh and Ganjali, 2012; Dai et al., 2014; Lee et al., 2014; Schmid and Suomela, 2013; Fang et al., 2015)
		Hybrid Distributed Controllers SDN (Fu et al., 2014)
	Fault-Tolerance	Hardware Redundancy (Berde et al., 2014; Ros and Ruiz, 2014; Najjar and Gaudiot, 1990; Chen et al., 2015; Kreutz et al., 2013; Botelho et al., 2013; Pashkov et al., 2014; Dorsch et al., 2016)
		Software Redundancy (Williams and Jamjoom, 2013; Katta et al., 2015; Heller et al., 2013; Reitblatt et al., 2013; Li et al., 2014b; Sgambelluri et al., 2013a; Astaneh and Heydari, 2016; Van Adrichem et al., 2014; Aguado et al., 2016)
		Information Redundancy (Borokhovich et al., 2014; Stephens et al., 2016; Chen et al., 2016a; Vissicchio et al., 2015)
	Flexibility	Programmable Data Plane (Song, 2013; Moshref et al., 2014; Mekky et al., 2014; Dixit et al., 2013; Martins et al., 2013; Zhang et al., 2013a; Bremler-Barr et al., 2016; Laufer et al., 2016; Bianchi et al., 2014; Cascone et al., 2015; Paolucci et al., 2019)
		Outsourcing Network Feature (Gibb et al., 2012; Chau et al., 2014; Wei et al., 2016; Melis et al., 2016; Syed and Van der Merwe, 2016)
	Elasticity	Programmable Data Plane (Yoshida et al., 2015; Xiong et al., 2018; Casellas et al., 2015; Sambo et al., 2014; Xie et al., 2016; Zhang et al., 2013b)
		Programmable Control Plane (Casellas et al., 2013, 2015; Sambo et al., 2014; Xie et al., 2016; Giorgetti et al., 2015; Paolucci et al., 2014; Cugini et al., 2016; Channegowda et al., 2013; Azodolmolky et al., 2011)

of users increased. A scalable network signifies a network that can scale or grow as the needs of the future. However, the scalability in the network adds some challenges such as communication overhead, processing capabilities, and resource constraints. The existing challenges and solutions for scalability of SDN are discussed in Section 3.1.

- **Fault-tolerance:** A network designed for availability is one that delivers consistent and reliable performance around-the-clock. In addition, the failure of a single link or a component of SDN should not significantly impact network performance (González et al., 1997). Network failure includes hardware faults, such as component malfunction, link failure; or software faults, such as, wrong routing decisions, loops, and black holes. A network needs to be designed that can cope up with faults and operate as required. The design of fault tolerant SDN for availability, utilizes redundancy at the cost of reduced performance or increased resources. Section 3.1 illustrates the challenges and solutions of the fault-tolerance in SDN.
- **Flexibility:** No matter how good the initial network design is, the available network staff must be able to manage and support the network. A network that is too complex or difficult to maintain cannot function effectively and efficiently is defined as the ability of network to adapt itself to the changes that occur in the network while it is in operation. These changes are sometimes predictable and can often occur unannounced. The network must be flexible enough to adapt and accommodate the future developments and the needs of the networks designers. The network is required to be designed such that it can modify its rules and behavior based on the dynamic variations occurring in the network. Section 3.2 describes the challenges and solutions for flexibility in SDN.
- **Elasticity:** Elasticity is the ability for dynamically adapting the workload changes inside the network. This dynamic adaptability refers to the maximum resources utilization during higher demand of network resources and reducing resource uses during lower demand of network resources. The reduction in resources during lower demand period will preserve the energy and reduce network maintenance cost while providing maximum resource utilization during peak demand will increase the quality of service. Elasticity will ensure the availability of resources at every instance of time irrespective of the demand (Dixit et al., 2014a). In the modern era of technical advancement, the optical fibre network speed is growing day-by-day to meet the growing demand of bandwidth. To maintain the elastic in optical fibre network, SDN is a vital solution due to its programmability of data plane and control plane that can handle variable network load. The detail description of high speed optical fibre network solutions using SDN for ensuring elasticity of the network is covered in Section 3.4.

2.2. Network implementation

Scalability, availability, and manageability are the main challenges in the network as shown in the previous section. A network designer collects the requirements of the users and designs a network that handles challenges. A network implementation is a phase which converses the network designed plan for a real world solution. While implementing an SDN in the real world, the challenges faced by network designers include integration of SDN with traditional or existing network, managing resources among devices, and managing virtual networks.

- **Integration with traditional network:** Despite the successes achieved in technology so far, SDN implementation is still in an early stage of development. Option available for transition is replacing the entire existing network infrastructure with the new SDN-enabled infrastructure. It is highly cost inefficient and usually not advisable. Techniques that can allow integration of the SDN with the existing traditional network are required. The existing challenges and solutions for integration of SDN with traditional network are discussed in Section 4.1.
- **Resource management:** Due to the large size networks, it is no longer practical to construct networks by connecting many standalone components without careful planning and design. Such components construct sub-networks in a given network. These sub-networks can either be physically separated or virtually distant. Consumable SDN resources include labels (*e.g.*, IP addresses, VLAN tags) and physical components (*e.g.*, switches, bandwidth, buffer space). Such resources need to be properly allocated and managed among various sub-network for proper operation and coordination among them. Section 4.2 describes the challenges and solutions of resource management in SDN.
- **Virtualization:** Network virtualization is the process of combining hardware and software components (*e.g.,* switches, controllers, other devices) and network functionality into a single, software-based administrative entity. The authors in Chowdhury and Boutaba (2009) define the SDN virtualization as the process of creating a software-based virtual network, by utilizing instances of hardware and software resources. The term virtual can be applied to applications, servers, resources, and the entire networks. Virtualization allows for efficient resource utilization in a network and provides sufficient isolation between different instances. All applications and policies in SDN are managed by a centralized controller. The problem of security and performance occurs due to sharing of same resources by centralized controller. Proper isolation among policies are required to the efficient virtualization schemes. The challenges and solutions in virtualization of SDN are discussed in Section 4.3

2.3. Network performance

Network performance refers to measures of QoS of a network as seen by the users. There are many different network parameters which are used to measure the performance of the network. Example of such parameters are latency, efficiency, and consistency. Before bringing into the actual implementation of the network, the performance of the network is usually anticipated by modeling and simulating the network architecture. The existing research work in this direction shows that the specialized architectures, hardware devices, and protocols are required to be designed to improve the performance of SDN.

- **Latency:** Latency of the network is defined as the interval between the cause and response observed in a system under monitoring. In packet switching network, latency is measured as the round trip delay time. The round trip time is preferred as it can be measured from a single point. Section 5.1 elaborates the challenges and solutions of latency in SDN.
- **Efficiency:** Efficiency of the network is a measure of how effectively the information is exchanged within the network. It mainly deals with forwarding rule storage and management, packet processing and forwarding in the network, and coordination between software and hardware switches to achieve high performance. Section 5.2 answers the question the following question: *How to improve the efficiency of an existing SDN?*.
- **Consistency:** Consistency of the network refers to coordination or harmony among different components of a complex system. A network is said to be consistent if at any given time the protocols and network state is constant through the network. In large size networks, due to communication delay among components, inconsistencies such as varying forwarding rules can arise. In Section 5.3, the challenges and solutions for consistency in SDN are discussed.
- **Traffic measurement:** The network management tasks such as anomaly detection, traffic engineering requires proper and efficient monitoring of the network. If the network is under operation then large number of packets flow can be observed at every instant. The monitoring of such packets in SDN requires a large number of resources of the data plane devices including CPU and memory. The existing challenges and proposed solutions for traffic measurement in SDN are discussed in Section 5.4.

2.4. Network verification

Data forwarding in networks is complicated and depends on proper functioning of multiple hardware and software entities. Building a large network is challenging and even more challenging is to build it without error. Considerable efforts are required for maintaining a network because errors and faults, such as loops, sub-optimal routing, black holes, etc. Even the networks that appear correct at static state might have subtle bugs that are required to be identified.

- **Testing:** Network testing is a process to check whether the actual results match the expected results and to ensure that the network system is defect free. Network testing is a software procedure and does not required any hardware component and therefore it reduces the cost of network verification. It helps to identify the defected components of the system. Various testing tools (*e.g.*, NICE, OFLOPS, FlowChecker, OFTEN) are needed for verification of a network before it can be brought up into the production. Section 6.1 illustrates the challenges and proposed solutions for testing the SDN.
- **Debugging:** Apart from testing, various faults or bugs can occur during the operational phase of the network. The complete network operation can be considered as software code and it requires various debuggers that can monitor the network behavior at run time. Debugging of SDN consists various challenges because of centralized control, large network size, and heterogeneity in network components. Section 6.2 illustrates existing challenges and solutions in debugging of SDN.
- **Security:** Security is a crucial feature for network verification, as it justifies the reliability of the network. Network security is the process of protecting the networking infrastructure, *i.e.* hardware and software, from unauthorized access, misuse, malfunction, modification, destruction, and improper disclosure. It includes activities that take preventive measures creating a secure platform for users and programs, thus maintaining the usability and integrity of the network. Network security includes measures such as access controls, firewalls, authentications, data loss prevention, and intrusion detection. Security challenges and solutions for SDN is elaborated in Section 6.3.

3. Network design

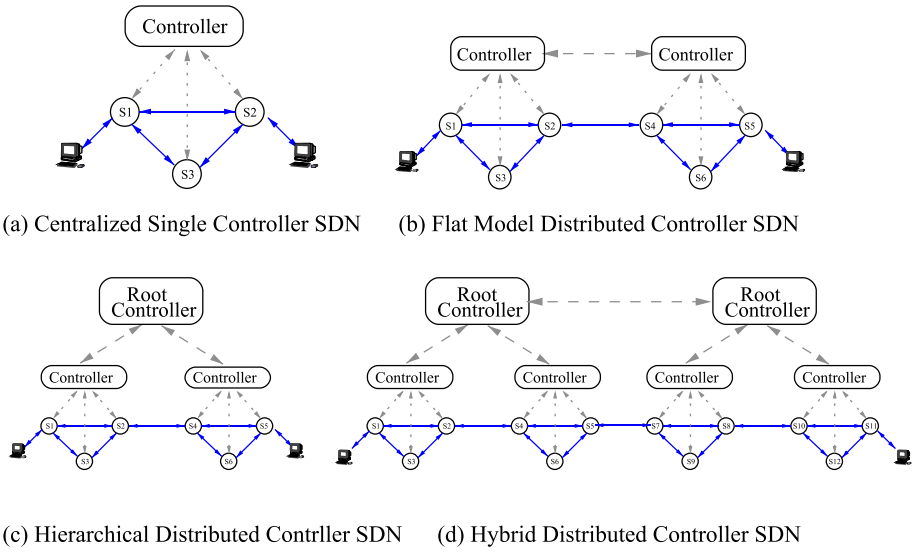
The SDN posed many open research issues regarding the designing of the network architecture. This section lists existing challenges and recent research work centered to deal with those challenges.

3.1. Scalability

Scalability in SDN is more stringent issue than the traditional network because of its centralized controller. The decoupled architecture of SDN requires a lot of communication messages between data plane and control plane for exchanging the information. A single controller based SDN architecture cannot handle the load when the network is scaled up. The existing work proposed various methods for handling the scalability issue in SDN. These methods are organized based on the number of controllers in the network, the connectivity between switch and controller, and the connectivity between controller and controller. Fig. 2 shows the different types of existing architectures of SDN in the literature. Table 3 summarizes the existing work related to the scalability in SDN.

3.1.1. Centralized single controller SDN

A single controller based SDN architecture consists only one controller for maintaining the global network state and controlling the entire data plan of the network as shown in part (a) of Fig. 2. Due to the bottleneck on a single controller, such architecture faces the scalability issues. The authors in Bifulco et al. (2016) and Curtis et al. (2011) show that the scalability issues in single controller architecture can be handled by delegating some control to the SDN switches. The control functionality is shared between the control plane and the data plane. Providing certain controlling ability to switches to reduce the workload of the controller and therefore improves the scalability of the network. While performing complex network operations, the controller needs to generate new packets frequently. It enhances the flexibility of the network. However, it also increases the load on the network and therefore restricts scalability of the network. In-Switch Packet generation (InSP) (Bifulco et al., 2016) delegated the control of the network by automatically handle the switch level packet generation. The switch level packet generation improves the performance of the network by avoiding the communication delay between switches and the controller. It also reduces packet generation time and network traffic. The authors in Devoflow (Curtis et al., 2011) illustrated that the traffic load on the controller can be reduced by handling shorter flows within the data plane. Flows are forwarded to the controller only if it cannot be handled by the data plane. These methods improved scalability but contradict the basic principle of SDN, *i.e.*, removing the controlling ability from the switches. Due to simplicity in designing, single controller based SDN can handle only a limited number of data plane elements and not suitable for heavy loaded network. The other limitation in such type of network is the single point failure, *i.e.*, failure of the controller shutdowns



(a) Centralized Single Controller SDN (b) Flat Model Distributed Controller SDN

(c) Hierarchical Distributed Contrller SDN (d) Hybrid Distributed Controller SDN

Fig. 2. Illustration of different SDN architectures.

the entire network. A distributed control plane based SDN architecture is proposed comprising of multiple controllers for handing the above limitations.

3.1.2. Flat model distributed controllers SDN

The Flat Model distributed Controllers SDN (FMC) architecture consists directly connected multiple controllers in SDN. Part (b) of Fig. 2 illustrates the block diagram of FMC architecture where switches are connected to controller and controllers are connected with each others. Each controller in the network maintains the global network information in a distributed way or without central control. The global network information is periodically synchronized using eastbound and westbound API. All the decisions in FMC architecture are made based on the global view of the network.

The distributed and connected controllers reduce the load on a single controller and help to improve the scalability of the network (Tootoonchian and Ganjali, 2010; Koponen et al., 2010; Yazici et al., 2014; Tam et al., 2011; Phemius et al., 2014; Dixit et al., 2014a). Each controller in the network manages the subnetwork lying in their domain. The subnetwork is abstracted as a node for higher layers. Global state information is communicated among the controllers using communication channels between controllers.

The authors in HyperFlow (Tootoonchian and Ganjali, 2010) proposed a logically centralized and physically distributed control plane SDN architecture. The controllers in the network maintain a consistent global view using event-driven mechanism. Flows among switches connected to a controller are locally handled, whereas a request for switches outside its domain is forwarded to the required controller. In Onix (Koponen et al., 2010), each controller maintains a Network Information Base (NIB) which stores the global network state. NIB of each controller is propagated and replicated among other controllers for achieving a uniform global state. ElastiCon (Dixit et al., 2014a) dynamically distributed the workload across the controllers by adding or removing controllers to the controller pool.

Maintaining global network state at each controller in flat model SDN requires lot of state update messages between the controllers. The problems of state consistency and lack of central authorization are common in this architecture. The other limitation is a large space requirement of the network. This is because each controller stores the same global state information. These problems can be overcome by designing a hierarchical distributed controllers SDN as it has a centralized body for storing global network information and managing controllers in the lower layers.

3.1.3. Hierarchical distributed controllers SDN

In Hierarchical Distributed Controllers SDN (HDC) architecture, the control plane comprised multiple layers (Hassas Yeganeh and Ganjali, 2012; Dai et al., 2014; Lee et al., 2014; Schmid and Suomela, 2013; Fang et al., 2015). The root controller presents at the top layer of the SDN and contains the global network view. The local controllers maintain the local network view and handle local applications. Local controllers communicate with the root controller for obtaining the required global information. Part (c) of Fig. 2 illustrates the block diagram of hierarchical distribution of controllers with one root controller and two local controllers. The local controllers perform processing near the switches for reducing the workload on the root controller. Local controllers periodically inform to the root controller about the local network state based on which a global state of the entire network is computed.

The authors in Kandoo (Hassas Yeganeh and Ganjali, 2012) proposed a local controller based distributed SDN firmware. The framework works between the control plane and the data plane of the SDN. A local controller handles local applications, whereas the root controller handles applications that require global view of the network such as load balancing, routing, etc. Scalability of distributed SDN can be improved by local operations efficiently. R-SDN (Dai et al., 2014) proposed a hierarchical distributed control plane architecture where each layer having a number of devices according to the Fibonacci series. The authors in IRIS (Lee et al., 2014) proposed a hierarchical architecture of SDN where the local controllers and the root controller together perform the network control operations. The local controllers periodically update the root controller about the local network information. The authors in Schmid and Suomela (2013) shown that the hierarchical distribution of controllers in SDN can also be used in the existing local algorithms for distributed computing to improve the performance of SDN. For example, a link assignment for the load balancing problem or a spanning tree problem can be explained in SDN for achieving the goal of a scalable network.

3.1.4. Hybrid distributed controllers SDN

The hybrid distributed controllers SDN architecture uses flat and hierarchical architectures for arranging the controllers in the network. Part (d) of Fig. 2 illustrates the hybrid distributed of controllers where local controllers are connected with root controllers and root controllers directly connected with each other. Multiple domain controllers, *i.e.*, root controllers in the network, maintain global network information

Table 3
Existing Challenges and proposed solution for scalability of SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Li et al. (2018)	QoS guarantee in satellite communication	QoS-oriented heuristic algorithms for routing	POX	OpenFlow	Prototype	Tr Constellation	Hybrid
Basta et al. (2017)	Minimizing network cost and data resources cost	Optimal placement of data center and NFV	–	–	Simulated	Mobile Core topology	Hybrid
Wu et al. (2018)	Cluster management and cluster security	Ant Colony Optimization	ONOS	OpenFlow	Simulated	–	Hybrid
Xie et al. (2017)	Coverage problem of controllers	Approximation approach to ILP	–	–	Simulated	Jellyfish	Hybrid
Zhou et al. (2017)	Transient anomalies, network updates	Lossless inter-domain route updates		OpenFlow	Simulated	–	Flat
Sallahi and St-Hilaire (2017)	Controller Placement Problem	Expansion model for CPP	–	–	Simulated	Random	Flat
Suh and Pack (2018)	Master controller assignment	Low complexity master assignment (LCMA)	–	OpenFlow	Simulated	Abilene	Flat
Blenk et al. (2016)	The optimal placement of the hypervisors	Mixed integer programming formulations	–	OpenFlow	Simulated	Abilene	Flat
Fu et al. (2015)	Path stretch problem	A hybrid distributed hierarchical control plane	Floodlight	OpenFlow	Simulated	Random	Hybrid
Li et al. (2014a)	Network security in control plane	Cost-efficient controller assignment algorithm	POX	OpenFlow	Simulated	–	Flat
Lange et al. (2015)	Controller placement problem	Heuristic approach	–	–	Simulated	Topology Zoo	Flat
Tuncer et al. (2015)	Network resource management	A placement algorithm	–	OpenFlow	Simulated	Abilene	Hybrid
Ji et al. (2014)	Energy consumption problem	SDN based control architecture	–	OpenFlow	Simulated	–	Hierarchical
Hu et al. (2015)	Unified big data platform for social TV	Cloud-centric platform with SDN support	Floodlight	OpenFlow	Testbed	–	Hierarchical
Tam et al. (2011)	Use multiple small independent controllers	Heuristic graph partitioning algorithms	–	–	Simulated	Rocketfuel	Flat
Lee et al. (2014)	The scalability and availability of the control plane	Recursive network abstraction via middle wares	Floodlight	OpenFlow	Simulated	–	Hierarchical
Phemius et al. (2014)	Single Point Of Failure in the control plane	Distributed control plane	Floodlight	AMQP	Simulated	–	Flat
Dai et al. (2014)	A scalable control plane	Network virtualization and fibonacci series	Opendaylight	–	Simulated	Random	Hierarchical
Fang et al. (2015)	Sclable hyper scale data center control plane	Small forwarding tables in nodes	–	–	–	–	Hierarchical
Fu et al. (2014)	The path stretch Problem in centralized abstracted control plane.	Hierarchical network views and routing method	Floodlight	OpenFlow	Simulated	Random	Hybrid
Tootoonchian and Ganjali (2010)	The long flow setup latencies in the switches	Distributed event-based control plane	NOX	OpenFlow	Simulated	–	Flat
Dixit et al. (2014a)	Uneven load distribution among the controllers	An elastic distributed controller architecture	NOX	OpenFlow	Simulated	Fat Trees	Flat
Koponen et al. (2010)	Turn networking problems into distributed systems problem	Flexible distribution primitives	NOX	–	Testbed	–	Flat
Hassas Yeganeh and Ganjali (2012)	Overhead of frequent events on the control plane	Offloading local event processing to local resources	–	OpenFlow	Simulated	Tree	Hierarchical
Schmid and Suomela (2013)	The distributed control of SDN networks and local algorithms	Distributed computing	–	–	Theoretical	–	Flat, Hierarchical
Curtis et al. (2011)	Costs in flow setups and statistics-gathering.	Wild-carded rules, allowing switches to make local decisions	–	OpenFlow	Testbed	Clos, HyperX	Single
Bifulco et al. (2016)	Offload some of controller tasks to data plane	Programmable in-switch generation of Packets	Ryu	–	Simulated	–	Single
Yazici et al. (2014)	Scalable and fault tolerant control plane	A distributed controller and an associated coordination framework	Beacon	OpenFlow	Simulated	–	Flat

and each domain controller is responsible for handling multiple local controllers. The authors in Fu et al. (2014) argued that the distributed control plane in hierarchical architecture increases the overall path length of the network when more layers are presented in the architecture. They proposed a hybrid architecture which classifies the layers in lower, middle and, top layers in SDN. The lower layer contains physical devices which are controlled by the local controllers. The local controllers are lying in the middle layers. The top layer consists of domain controllers which are administered the area controllers as physical devices. The domain controllers maintain a uniform global network state, by coordinating among themselves at regular intervals. Link discovery, host management, topology management, routing, and storage models of SDN are run both at the local layer and layer of local and domain controller. Optimized routing is also used on above both layers and top layer. The global shortest routes are used on the top layer for forwarding the information.

3.2. Fault-tolerance

An SDN architecture consists of separate control plane and data plane. The control plane solves the problem of forwarding the packets from source to destination. The data plane takes the responsibility of physical forwarding of packets onto the wire. By separating of control plane and data plane, the SDN allows the controller to make off-the-wire routing decisions and enable network administrators to properly configure true and intelligent load-balancing. Since, the control plane in SDN works as the brain of the network and therefore fault at the control plane affects the working of the entire network. Usually, the main reason of fault at the control plane is the failure of controller(s) in SDN. The data plane consists of switches and handles the communications between switches and switch to controller. The fault at data plane arises due to the problem in switches or the links between switch-to-switch or switch-to-controller. The failure of control plane or data plane reduces the availability of the network. As discussed earlier, redundancy is the most common means for achieving a fault tolerant network. Redundancy uses extra components in the system so the system was not affected at the time of failure. Table 4 presents the summary of the fault-tolerant literature in SDN. In this section, we discuss how to handle fault-tolerant challenge in SDN.

3.2.1. Hardware redundancy

The hardware redundancy handles the fault tolerant issues in SDN by using additional hardware that can be used at the time of failure. The SDN installs the same copy of controller program in different systems to achieve hardware redundancy (Berde et al., 2014; Ros and Ruiz, 2014; Najjar and Gaudiot, 1990; Chen et al., 2015; Kreutz et al., 2013; Botelho et al., 2013). However, these systems need to maintain a centralized view of the network.

The authors in Berde et al. (2014) use Open Network Operating System (ONOS) to design a fault tolerant SDN architecture. The architecture uses graph based models for managing multiple Floodlight controllers. The architecture obtained fault tolerance by linking each switch with multiple controllers. Data among all controllers was periodically synchronized. At a time, only one controller acted as master and the other controllers would take control in case of any fault occurrence. The main limitations of the proposed work are high latency and storage requirement for maintaining consistency in SDN. The authors in Ros and Ruiz (2014) proposed a solution for solving fault-tolerant controller placement problem. It determined the location of controllers that minimized the access time for switches. The solution first used a graph structure to represent the overall topology along with the possible locations of the controllers. So, each switch is connected with multiple controllers for achieving fault tolerance. Next, calculate the cost based on delay of each link in the graph. Finally, network reliability is calculated based on the probabilities maintained for each node and controller reliability. Since computing an overall network reliability over a threshold is an NP hard problem and therefore a lower bound solution is obtained.

A highly available controller architecture is proposed in Pashkov et al. (2014) for fast failure recovery. This high availability is due to the presence of a highly available control plane. This approach is beneficial for enterprise networks, where high availability of the system is mandatory despite of frequent failures. If the failure persists for a longer run than it will reduce overall profit of an organization. Authors in Dorsch et al. (2016) made a complete analysis of local and central mechanism for faster failure recovery in SDN based smart grid. The failure detection using the controller is beneficial in terms of optimal path finding due to the presence of the global view. The smart grid operation will be hindered, if the delay of failure recovery is high.

A cluster based hardware redundancy method, known as virtual controller redundancy, is proposed in Najjar and Gaudiot (1990). The method forms a cluster of controllers where a controller act as master for serving the network and remaining controllers acting as slaves. The slave helps to maintain their consistent global network state. When the master controller fails, then one of the slave controllers within the cluster works as new master to take over the network operation. From view of the data plane, only single virtual controller exits and perform controlling operation at all times. Hardware redundancy uses additional hardware thus demand additional network resources and increases network cost. To reduce the cost of the network, some authors proposed software redundancy methods instead of hardware redundancy.

3.2.2. Software redundancy

The software redundancy uses multiple versions of a software in the system and assumes that different versions of the software will not fail at the same time instant. Fault tolerance can be achieved by using software redundancy in the network. The most common approach to achieve the software redundancy is by using the replicas of different applications and controller program (Williams and Jamjoom, 2013; Katta et al., 2015; Heller et al., 2013). In this approach, if a replica encounters a fault, then another replica can take over the work. Another approach is designing a controller code using high level language that contains modules for achieving fault tolerance (Reitblatt et al., 2013; Li et al., 2014b).

Authors in Williams and Jamjoom (2013) discuss about the processing delay and loss of information when a controller fails and other replica takes over the SDN. The authors proposed a system, called RuleBricks, that collects the details about flow rules of the controller and designs the backup rule set in case any failure occurs. The system reduces the space complexity by maintaining only the hierarchical wild card rules of OpenFlow in the switches. Katta et al. (2015) proposed a framework to achieve an abstraction of a fault-free network. It handles the controller consistency and entire operation cycle as a single transaction. The operation cycle includes event initiation and command execution on the switches, event processing on the controllers. It extends the concept of replication of state machines by including switch states along with the controller states. A high level language, called FatTire, was proposed in Reitblatt et al. (2013) for writing fault-tolerant network programs. The language uses regular expressions to specify the forwarding paths and fault-tolerance requirements. While programming for each forwarding path based on a policy. An alternate or backup path can be installed on the switches. The compiler converts these regular expressions to the flow rules that can be installed on the switches.

In Sgambelluri et al. (2013a), authors emphasized on the effective mechanism of fast recovery on OpenFlow architecture. The authors introduced a novel mechanism for maintaining backup flow with different priorities and ensure efficient network utilization. Authors in Astaneh and Heydari (2016) presented a failure restoration technique for minimizing the recovery cost. Lowest possible operations similar to that of Dijkstra algorithm are used to optimize recovery time of failure. Authors in Van Adrichem et al. (2014) demonstrated that reactive and

Table 4
Fault-Tolerant in SDN and its existing challenges and solutions.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Wu et al. (2018)	Cluster management and cluster security	Ant Colony Optimization	–	OpenFlow	Simulated	–	Hardware redundancy
Xie et al. (2017)	Coverage problem of controllers	Approximation approach to ILP	–	–	Simulated	Jellyfish	Hardware redundancy
Wang et al. (2018)	Control-plane communication overhead/delay	Flow redirection	Opendaylight	OpenFlow	Testbed	–	Hardware redundancy
Song et al. (2017)	Control path reliability problems	Control path reliability management framework	Floodlight	OpenFlow	Simulated	–	Hardware redundancy
Zhang et al. (2017)	Link failures in data plane	Local fast reroute (LFR) algorithm	Floodlight	–	Simulated	USNET	Software redundancy
Caria et al. (2016)	Study of hybrid networks	New hybrid network architecture	–	–	Simulated	Cost266, Nobel-EU	Hardware redundancy
Li et al. (2014a)	Network security in control plane	Cost-efficient controller assignment algorithm	POX	OpenFlow	Simulated	–	Hardware redundancy
Lange et al. (2015)	Controller placement problem	Heuristic approach	–	–	Simulated	Internet Topology Zoo	Hardware redundancy
Berde et al. (2014)	Fault tolerant SDN architecture	Redundant controller	Floodlight	OpenFlow	Prototype	Internet2	Hardware redundancy
Ros and Ruiz (2014)	Controller placement problem	Max-flow, Disjoint Path	–	–	Simulated	Internet Topology Zoo	Hardware redundancy
Williams and Jamjoom (2013)	Faults in control plane	Chord Assignment, Distributed hash table	–	OpenFlow	Simulated	–	Software redundancy
Stephens et al. (2016)	Scalability of fast failover schemes	Compression aware routing	–	–	Simulated	EGFT, Jellyfish	Information redundancy
Reitblatt et al. (2013)	Fault tolerant SDN	Modular language	–	OpenFlow	Prototype	–	Software redundancy
Borokhovich et al. (2014)	Computing failover tables	Modulo search, DFS, BFS	–	OpenFlow	Theoretical	–	Information redundancy
Katta et al. (2015)	Fault tolerant control plane	Single transaction scheme	Ryu	OpenFlow	Simulated	–	Software redundancy
Najjar and Gaudiot (1990)	Fault tolerant control plane	Virtual controller redundancy	–	–	Simulated	Torus, Binary n-cube	Hardware redundancy
Chen et al. (2016a)	Fault tolerant SDN	Protection based recovery method	Ryu	OpenFlow	Simulated	Grid	Information redundancy
Sgambelluri et al. (2013a)	Fast recovery in OpenFlow network	Architecture for support segment protection	NOX	OpenFlow	Emulated	Ring	Software redundancy
Astaneh and Heydari (2016)	Restoration in multiple failure	Few operation low cost recovery path	–	OpenFlow	Simulated	USnet, ERnet	Software redundancy
Pashkov et al. (2014)	Fast failure recovery	High available controller	NOX	OpenFlow	Simulated	–	Hardware redundancy
Van Adrichem et al. (2014)	Non-trivial failure recovery	Bi-directional forward detection mechanism	–	OpenFlow	Prototype	SURFnet, NREN	Software redundancy
Dorsch et al. (2016)	Fast failure recovery	Failure detection using controller	–	OpenFlow	Simulated	IEC 61850	Hardware redundancy
Aguado et al. (2016)	Failure and congestion detection in optical network	Archtecture for reconfiguring network services	ABNO, POX	OpenFlow	Simulated	–	Software redundency

proactive mechanism of failure recovery in SDN is having a long delay. It involves three steps: failure detection, communication with the controller, and computation of the shortest path. Authors thus proposed a bidirectional forward detection mechanism for faster failure recovery.

3.2.3. Information redundancy

Information redundancy provides more information for handling the fault in SDN. Link failures in data plane causes switches to access the controller for defining a new rule. This leads to increased in latency and communication overhead. Backup routes technologies install the supplementary routing information within the flow-tables of the switches. It also includes the required forwarding rules for achieving fast fail-over (Borokhovich et al., 2014; Stephens et al., 2016; Chen et al., 2016a; Vissicchio et al., 2015). Such local fast fail-over mechanisms use the information redundancy to handle the failure problem.

Alternate rules generated for fail-over mechanism in the switches must guarantee the connectivity in the SDN. The authors in Borokhovich et al. (2014) uses group tables of OpenFlow as alternate tables for handling link failure. Such group tables work as information redundancy in SDN. In the proposed fail-over mechanism, tables are grouped into buckets. This mechanism chooses a bucket at a time for forwarding the packet. Installing backup routes, however requires a large Ternary Content Addressable Memory (TCAM) space on the switches which usually not available. We can solve such problems by compressing the information. A compression method that utilizes compression aware routing scheme is proposed in Stephens et al. (2016). The scheme accommodates the rules in the limited TCAM space of switches. It uses wild card rules to compress multiple rules with the same output and action. Another memory efficient fault tolerant approach is proposed in Chen et al. (2016a) using VLAN tags. Instead of each flow backup rules are computed based on each link. A VLAN tag is provided for each link identification while defining backup rules.

3.3. Flexibility

Network flexibility incorporates the ability of the network to adapt the variation in the network condition. The SDN performs various operations simultaneously, such as forwarding packets, security, isolation, address translation, etc. The performance of the SDN can be improved by outsourcing certain network features and operations to external providers. It can also be improved by transfer of some control features from the controller to the switches. However, such flexibilities in SDN add challenges in the network. Table 5 presents the summary of the literature related to flexibility in SDN.

3.3.1. Programmable data-plane

In case of dynamic applications, varying network traffic and changes in the network topology, flow rules are required to be re-computed by the controller. To increase flexibility of the network, some decision ability can be provided to the switches. It helps to reduce controller overhead and decreases response time (Song, 2013; Moshref et al., 2014; Mekky et al., 2014; Dixit et al., 2013). Along with the switches, the data plane also consists of middle-boxes which are required to be effectively managed by the controller to make the network more flexible (Martins et al., 2013; Zhang et al., 2013a; Bremler-Barr et al., 2016; Laufer et al., 2016).

Protocols in SDN have specific flow rules for matching the entries and required actions. Authors in Bosshart et al. (2013) proposed a new RISC inspired pipelined architecture for reconfiguring the match tables of the switches. It provides a set of pipelined flow tables with variable size. The match field definitions can be altered based on policy requirement. It allows arbitrary modified packets to be handled in the data plane. The main challenge in this approach is the fabrication of new chips for the switches. The authors in Song (2013) shown that the switches in SDN can be made programmable similar to the Central Processing Unit (CPU). As the CPU has very limited instruction set, yet

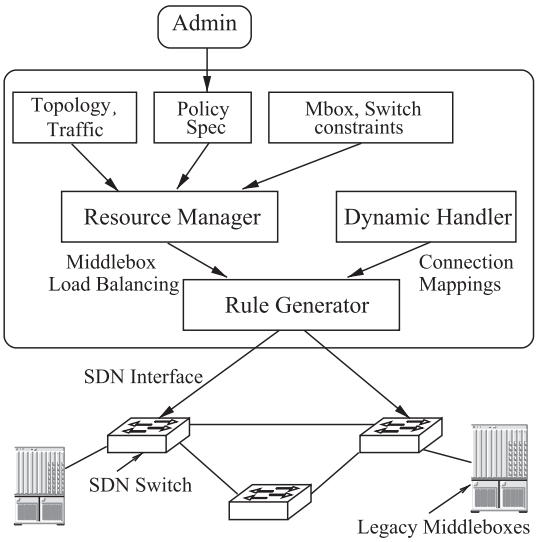


Fig. 3. Configuring data plane comprising of switches and middle-boxes (Qazi et al., 2013).

can perform all complex operations. Similarly, switches must be able to forward any packet without prior knowledge of policies. The packet processing in CPU is however slow. Thus another solution is adding the CPU to the existing switch (Mekky et al., 2014) where the most of the forwarding could be handled by the hardware switch.

The authors in Martins et al. (2013) proposed an approach for converting the middle-boxes to software programs. The approach can be dynamically controlled and instantiated as and when required. The software interface designed for middle-boxes must be faster, smaller in size, and flexible. However, the designed middle-box software cannot provide similar performance as its hardware counterpart. Another approach, simplifying implementation of middle-box was proposed in Qazi et al. (2013). The authors treated middle-boxes as black-boxes and eliminated the requirement of human intervention in middle-box placement. The main objective of this work is to simplify middlebox traffic steering. Fig. 3 shows the block diagram of the architecture proposed in Qazi et al. (2013). It shows that the inputs needed for various components, the interactions between the modules, and the interfaces with the data plane. The architecture needs to configure the switches; middleboxes do not need to be extended to support new SDN-like capabilities. The resource manager takes into account both middlebox and switch constraints in order to optimally balance the load across middleboxes. The dynamic handler model infers, mapping between the incoming and outgoing connections of middle-boxes. The Rule generator model ensures that middleboxes with stateful session semantics will receive both the forward and reverse directions of the session.

The paper (Zhang et al., 2013a) presented a framework for finding the best positions of middle-boxes to connect them in SDN. Authors in Bremler-Barr et al. (2016) proposed a software defined architecture of middle-boxes, called OpenBox. It decouples the control plane of network-functions from data plane. It also allows reuse of data plane elements by multiple logical network function. Fig. 4 illustrates the architecture of OpenBox. It provides greater flexibility in terms of network functions development and its deployment, multi-tenancy support with alternate or backup path is specified that can be used in case of a failure.

The authors in Bianchi et al. (2014) suggested that in SDN a prior assumption of using dummy switches at data plane and a centralized controller for handling all the control flow is not a wise choice. The authors thus proposed a scheme of low-level programming at dummy switches such that some of the flow control decisions can use local state information. This processing at switches will reduce the number of a

Table 5
Existing Flexibility Challenges and its Solution in SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Bruschi et al. (2017)	Future Internet personal cloud services	SDN with fog-computing approach	–	–	Simulated	–	Programmable data plane
Zamani et al. (2017)	Leverage computational capabilities of the network	Prioritize workload processing depending on data.	POX	OpenFlow	Simulated	–	Programmable data plane
Bao et al. (2018)	Traffic imbalance due to Node mobility	Supervised learning for prediction	NS-3	–	Simulated	Random	Programmable data plane
Aujla et al. (2018a)	Handling huge data in Dcs	Workload Slicing scheme	–	OpenFlow	Simulated	–	Programmable data plane
Luo et al. (2016)	Multi stage attack	Attack mitigation mechanism	–	–	Simulated	–	Outsourcing network
Anadiotis et al. (2016)	Support big data processing in WSN	SDN-WISE protocol	–	–	Simulated	Ring, Grid	Programmable data plane
Gharakheili et al. (2016)	Bandwidth bottleneck in access networks	Two-Sided Resource Management	Floodlight	OpenFlow	Simulated	–	Programmable data plane
Li et al. (2016a)	Changes in REST APIs in SDN	PetriNet-based REST service	SOX	OpenFlow	Simulated	–	Northbound APIs
Rahman et al. (2016)	Network virtualization in wireless	A multi-criteria utility model	–	–	Simulated	–	Programmable data plane
Kuang et al. (2016)	Extracting heterogeneous, core IoT data	Tensor based SDN-IOT	–	OpenFlow	Simulated	–	Programmable data plane
Faraci and Schembra (2015)	Orchestration and management of SDN	A green management policy	–	OpenFlow	Simulated	–	Programmable data plane
Wood et al. (2015)	Achieving software-based network	Integrate SDN with NFV	–	–	Theoretical	–	Programmable data plane
Jo et al. (2014)	Multi-home networking control	In-home consumer electronic devices	–	–	Prototype	–	Programmable data plane
Abolhasan et al. (2015)	Scalability challenge in wireless distributed networks	Allow nodes to make distributed routing decisions	–	–	Simulated	–	Programmable data plane
Ding et al. (2015)	Integrating service chaining with SDN	Using service chain as a service in SDN	–	–	Theoretical	–	Outsourcing network
Song (2013)	Limited programmability	Programmable switch	Floodlight	OpenFlow	Simulated	–	Programmable data plane
Zhang et al. (2013a)	Traffic diversion through middleboxes	Middleboxes placement	NOX	OpenFlow	Testbed	–	Programmable data plane
Martins et al. (2013)	Programmability of data plane	Programmable middle-box	–	–	Testbed	–	Programmable data plane
Gibb et al. (2012)	Outsourcing in-network features	resource outsourcing	Beacon	OpenFlow	Testbed	–	Outsourcing network
Wang et al. (2012)	Combining SDN with big data	Hadoop and optical switch	–	OpenFlow	Simulated	–	Outsourcing network
Wei et al. (2016)	Security of outsourced middleboxes	Outsourced firewall	–	–	Testbed	–	Outsourcing network
Melis et al. (2016)	Privacy of network policies	Cryptographic approach	–	–	Simulated	–	Outsourcing network
Jyothi et al. (2015)	Path diversity for network fabrics	Source routing	–	OpenFlow	Simulated	Leaf Spine, Fat Trees	Outsourcing network
Bremler-Barr et al. (2016)	Management of network functions	SDN framework for Nfs	Opendaylight	OpenFlow	Testbed	–	Programmable data plane
Qazi et al. (2013)	Middlebox traffic steering	Logical middlebox routing policy	POX	OpenFlow	Simulated	Internet2	Programmable data plane
Bianchi et al. (2014)	Distributed decision in SDN	Flow control upon local decision	–	OpenFlow	Prototype	–	Programmable data plane
Cascone et al. (2015)	Stateful data plane	Local decision making	–	OpenFlow	Simulated	–	Programmable data plane
Paolucci et al. (2019)	Programming data plane structure and behavior	Protocol independent packet processors	–	OpenFlow	–	–	Programmable data plane

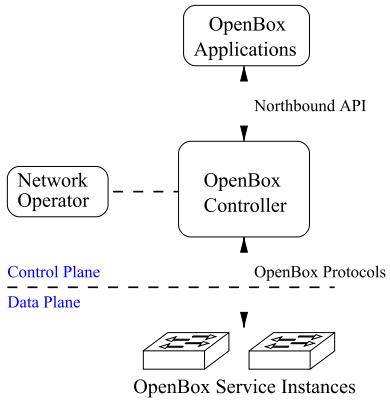


Fig. 4. An OpenBox architecture proposed in Zhang et al. (2013a).

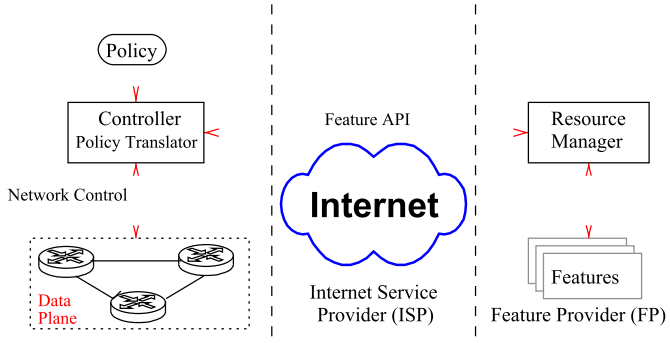


Fig. 5. Outsourcing to feature provider in SDN (Gibb et al., 2012).

flow control messages sent to the controller. Similarly, in Cascone et al. (2015) authors emphasized on the OpenFlow abstraction at data plane for local decision making. Authors present two different traffic management applications for justifying the local decision making at data plane. A prototype is also developed for experimental analysis for verifying the results. Further, in Paolucci et al. (2019) authors used the data plane programming for utilizing the potential of protocol-independent packet processors programming language. The language will help in a multi-layer architecture programming with edge segmentation. If the forwarding procedure in SDN involves network state and profile variation using history of flow statics, then there is additional latency at data plane. To overcome such latency authors suggested the use of packet processors programming language for programming data plane structure and behavior.

3.3.2. Outsourcing network features

Flexibility in an SDN can be increased by outsourcing the network functionality to an external agent or module (Gibb et al., 2012; Chau et al., 2014; Wei et al., 2016; Melis et al., 2016; Syed and Van der Merwe, 2016). In outsourcing network features, all operations need not be performed within the network. Thus the SDN can provide all facilities to users and still remain simple. It also helps to incorporate the recent growing technologies with SDN to achieve better application plane and control plane interaction with the network adaptability. An example of such techniques are big data, cloud computing, optical switches, etc.

The authors in Gibb et al. (2012) presented an architecture for adding functionality to networks using outsourcing. Fig. 5 illustrates the components of the architecture, *i.e.*, policy translator, feature API, controller, and resource manager. The policy translator combines all features and defines related traffic rules based on the combined policies. Resource manager creates, deletes, and monitors the movement of feature instances. The main advantages of this work are reduced cost and management complexity, improved features through outsourcing specialization, and increased option in services. The authors also observed the performance degradation in terms of increased latency and security issues. Authors in Wang et al. (2012) adopted a similar approach and used Hadoop, a big data application, to design a cross-layer SDN control plane. Cross-layer SDN control plane provides an enhanced inter-layer interaction inside the SDN. The control plane can dynamically configure the network at run time. It is a centralized scheme for managing the jobs. A job tracker node manages the scheduling and requirement of each job. The proposed approach used MEMS optical switches to perform better in a dynamic environment. A MEMS optical switch improves the network switching time. The security issues in outsourcing network functionalities are discussed in Melis et al. (2016). The authors used cryptographic approaches for ensuring privacy in functions provided via outsourcing.

Scalable high-performance data plane with a limited flow-table can also be achieved by using fabric scheme, where complex application-sensitive functions are factored out (Casado et al., 2012; Jin et al., 2016). The core network is simple and has fixed paths per-established in the flow table. The edge nodes are capable of performing complex operations. All decisions in fabric architecture had taken by the edge nodes based on available resources and current network traffic (Jyothi et al., 2015). The core network switches reduce the storage space requirement by storing only one rule per hop and port pair. As the network diameter increases, the number of rules increase but there will be no increase in the number of switches. A core network switch supports only source-based routing, which provides static forwarding tables and maximum path utilization. The link failure and its response conditions were however not discussed.

3.4. Elasticity

This section describes the issues and challenges in high speed optical fibre network and their solution using SDN. Elasticity in high speed optical fibre network using SDN is growing fast due to its advantage of programmable control plane and data plane. Table 6 illustrates the various literature using SDN for maintaining elasticity in an optical fibre network. The work on SDN based elasticity in optical fibre network is mainly divided into two parts: programmable data plane (Yoshida et al., 2015; Xiong et al., 2018; Casellas et al., 2015; Sambo et al., 2014; Xie et al., 2016; Zhang et al., 2013b) and programmable control plane (Casellas et al., 2013, 2015; Sambo et al., 2014; Xie et al., 2016; Giorgetti et al., 2015; Paolucci et al., 2014; Cugini et al., 2016; Channegowda et al., 2013; Azodolmolky et al., 2011).

3.4.1. Programmable data plane

The elasticity in an optical fibre network can be achieved by programming the switches at data plane. Data plane programming for the elastic optical fibre network is discussed in this subsection with their benefits and shortcoming in various literature.

The authors in Zhang et al. (2013b) determined the problem of non-elasticity of the current optical fibre network. To overcome such limitations in the network, the authors proposed an OpenFlow based flexi-grid optical network model. It is a software programmable based feature that will ensure elasticity. The experimental analysis of the paper shows that there must be a provision of a dynamic light-path establishment for achieving elasticity in the optical fibre network. An application based architecture using SDN is proposed in Yoshida et al. (2015). This architecture is used for provisioning end-to-end optical fibre based transport services with supporting multi-layer, multi-technology network scenarios.

The flexi-grid elastic optical network issues are discussed in Xiong et al. (2018). The authors presented a complete description of the failure issues in optical network and also provide a solution through SDN based triggered precomputation. This precomputation is used for com-

Table 6
Existing Elasticity Challenges and its Solution through SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Zhang et al. (2013b)	Non-elastic optical fibre network	OpenFlow based flexi grid	–	OpenFlow	Emulated	Random	Programmable control plane
Sambo et al. (2014)	Elasticity of transponder, code, and filter configuration	OpenFlow architecture to control high speed transmission	–	OpenFlow	Simulated	–	Programmable data plane, control plane
Giorgetti et al. (2015)	Dynamic restoration in transport network	Restoration for elastic optical network	C++ simulator	OpenFlow	Simulated	Pan-European	Programmable control plane
Paolucci et al. (2014)	Restoration capability in transponder	SDN based architecture for SBVTs	Flex controller	OpenFlow	Simulated	TEL, BT	Programmable control plane
Cugini et al. (2016)	Effective configuration of transmission parameter	Reoptimisation of high capacity frequency slot	–	OpenFlow	Simulated	–	Programmable control plane
Yoshida et al. (2015)	Elastic bandwidth slicing provisioning	SDN based application architecture	ABNO controller	OpenFlow	Simulated	–	Programmable data plane
Xiong et al. (2018)	Elastic optical network failure issues	SDN based triggered precomputation	Floodlight	OpenFlow	Simulated	NSFNet, COST239	Programmable data plane
Channegowda et al. (2013)	Operating heterogeneous devices in optical network	SDN based solution for seamless operation	NOX	OpenFlow	Simulated	–	Programmable control plane
Azodolmolky et al. (2011)	End-to-end delivery of packet	SDN packet and optical switching	GMPLS controller	OpenFlow	Simulated	–	Programmable control plane
Casellas et al. (2013)	Elastic optical network requirement	Path computation description	POX	OpenFlow	Simulated	14 node JT	Programmable control plane
Casellas et al. (2015)	Heterogeneous interconnection	Protocol architecture based on abstraction	GMPLS controller	OpenFlow	Simulated	14 node JT	Programmable control, data plane
Xie et al. (2016)	Energy consumption in DCN	Elastically controlling the resources of DCN	GMPLS controller	OpenFlow	Simulated	Fat-tree, BCube topology	Programmable control, data plane

Table 7
Network implementation challenges in SDN.

Network Implementation	Integration with Traditional Network	Implementing SDN using Traditional Network (Hand and Keller, 2014; Pan et al., 2013; Nelson et al., 2015) Implementing SDN using Hybrid Network (Sgambelluri et al., 2013a; Guo et al., 2014; Hong et al., 2016; Sieber et al., 2016; Wang et al., 2017a; Jin et al., 2017; Xu et al., 2017a; Sinha et al., 2017; Wang and Huang, 2016; Das et al., 2010)
	Resource Management	Resource Management in Control Plane (Hassas Yeganeh and Ganjali, 2012; Dai et al., 2014; Dixit et al., 2014b; Baldin et al., 2014) Resource Management in Data Plane (Moshref et al., 2013, 2015; Ferguson et al., 2012; Pereíni et al., 2014)
	Visualization	One-to-One Mapping (Belbekkouche et al., 2012) All-to-One Mapping (Kang et al., 2013; Soulé et al., 2014) One-to-Many Mapping (Lin et al., 2012; Al-Shabibi et al., 2014; Ghorbani and Godfrey, 2014; Gutz et al., 2012; Monsanto et al., 2013)

puting backup paths and helps in maintaining elasticity in communication among devices connected through the optical fibre. The interconnection of widely spread data centres has provided end-to-end connectivity to users. This heterogeneous interconnection issues are discussed in Casellas et al. (2015). It is not only due to data transmission but also depends upon the deployed data plane and control plane. The authors defined a protocol architecture for a heterogeneous network based on abstraction and control.

3.4.2. Programmable control plane

Similar to the data plane programmability the SDN control plane programmability is also used to provide elasticity in an optical fibre network. This subsection will discuss various work in the field of control plane programming for achieving elasticity in optical fibre network.

For high-speed optical fibre network connections, authors in Sambo et al. (2014) proposed an SDN based controller that is capable enough to set the parameter like code rate at both data plane and control plane. The controller will manage the high throughput requirement of high-speed optical network. The authors presented an extension of an OpenFlow based architecture to control high-speed channel on time-frequency packaging. In Giorgetti et al. (2015) authors proposed elastic optical network restoration based on SDN. Through simulation analysis by programming the control plane the authors show that recovery time in the transport network is significantly reduced. For achieving relevant restoration capability authors in Paolucci et al. (2014) proposed SDN architecture that can adequately handle sliceable bandwidth. In this, an integer linear programming model is proposed for adapting with the transmission capacity of the traffic.

An effective configuration of transmission parameters like modulation error codes, spacing of carries for the ultra high-speed channel is proposed in Cugini et al. (2016). It improves network utilization. The authors re-optimized the frequency slot using SDN. The authors in

Table 8
SDN challenges on Integration with Traditional Network and its existing solutions.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Tanha et al. (2018)	Progressive Migration to SDN	Ant Colony Optimization Rank-based algorithm	–	–	Simulated	Internet Topology Zoo	Hybrid Network
Xu et al. (2018)	Scalability challenge in SDN	Hybrid switching (HS) design	–	OpenFlow	Testbed	HyperX	Hybrid Network
Wang et al. (2016a)	Finding minimum-power network subsets	Heuristic algorithm	–	–	Simulated	Fat-Trees	Hybrid
Caria et al. (2016)	Study of hybrid networks	New hybrid network architecture	–	–	Simulated	Cost266, Nobel-EU	Hybrid
Salsano et al. (2016)	Integrating SDN in IP networks	Open Source Hybrid IP/SDN networking	–	–	Testbed	–	Hybrid
Kar et al. (2016)	Budgeted Maximum Coverage Problem	Heuristic solutions	–	–	Simulated	–	Hybrid
Hand and Keller (2014)	Mimic SDN control in traditional network	Use capabilities of openflow	–	OpenFlow	Testbed	–	Traditional network
Pan et al. (2013)	Incompatibility of flow table processing and controller	Rule conversion SDN to hardware rules	–	OpenFlow	Simulated	–	Traditional network
Nelson et al. (2015)	Migration from existing network to SDN	Policy definition	–	OpenFlow	Simulated	–	Traditional network
Sinha et al. (2017)	SDN deployment in hybrid MPLS network	MPLS routing in hybrid network	Ryu	OpenFlow	Simulated	Mesh	Hybrid
Hong et al. (2016)	SDN deployment in existing network	Incremental Hybrid network deployment	–	–	Simulated	ISP, ENTR	Hybrid
Jin et al. (2017)	Migration from existing network to SDN	MAC address based forwarding	Magneto	OpenFlow	Testbed	–	Hybrid
Xu et al. (2017a)	Deployment of Hybrid SDN	DFS, randomized routing mechanism	–	OpenFlow	Simulated	Monash University	Hybrid
Wang and Huang (2016)	SDN based smart grid communication	Aggregation point planning	–	–	Simulated	–	Hybrid
Guo et al. (2014)	Partial SDN deployment	Fixed weight setting in OSPF nodes	–	–	Simulated	Rocketfuel	Hybrid
Sgambelluri et al. (2013a)	Core and metro network scenarios	Comprehensive control of coverage	–	OpenFlow	Emulated	Metro Ethernet ring	Hybrid
Das et al. (2010)	Trade-off between IP network and transport network	Unified control plane	NOX	OpenFlow	Simulated	–	Hybrid

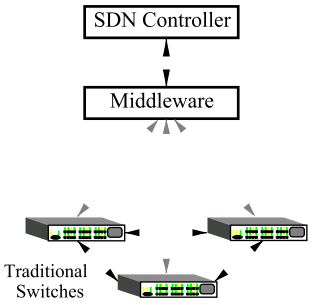


Fig. 6. Integration of SDN with traditional networks.

Channegowda et al. (2013) proposed an SDN based solution for seamless operation of an optical fibre network. The devices in the network are heterogeneous and require elasticity of network for smooth functioning. The authors demonstrated the result for OpenFlow protocol for flexible DWDM grid transport technology. Authors in Azodolmolky et al. (2011) studied the end-to-end delivery of the packet. It involves network control, operation, and management of the packet in the network. Authors proposed an elasticity solution based on SDN for packet and optical circuit switching domain. In this, there is an integration of OpenFlow and GMPLS control plane. The authors in Casellas et al. (2013) addressed the requirement of elastic optical fibre network and high system performance. Here, authors presented a detailed description of control plane for the path computation element.

4. Network implementation and evolution

This section describes the issues and challenges in the deployment of SDN, the distribution of the available resources, and management of the virtual networks, as shown in Table 7.

4.1. Integration with traditional networks

The main challenge in the deployment of SDN is to replace the existing network hardware with the OpenFlow enables devices. The literature in SDN uses traditional hardware and hybrid network approaches for integrating the SDN components (*e.g.*, openflow switches and controller) with traditional network. Table 8 presents existing work on SDN challenges on Integration with Traditional Network and its solutions.

4.1.1. Implementing SDN using traditional network

The approaches (Hand and Keller, 2014; Pan et al., 2013; Nelson et al., 2015) use an SDN controller for controlling traditional switches as shown in Fig. 6. It does not require high cost OpenFlow switches and therefore helps to reduce the network cost.

The authors in Hand and Keller (2014) proposed a system, called ClosedFlow, which incorporates techniques for exercising SDN control over existing proprietary hardware. The proposed system uses a minimal instance of the Open-Shortest-Path-First (OSPF) routing protocol for establishing a control channel. This channel will provide communication between the controller and the traditional switches. Running OSPF provided information about links among the switches. VLAN channel was used for communication with the controller using remote access protocols such as Telnet or SSH. Each switch collects information about its neighbors for topology discovery and used remote logging in the facility to inform the controller. Using information from switches, the controller can create a global network view. Packet matching and actions taken on the packet was achieved using Access Control Lists (ACL) and the route maps at each switch. The system closely mimics the fine grain control available in the OpenFlow.

The authors in Pan et al. (2013) proposed an innovative middle layer called FlowAdapter. It converts flow rules from SDN controller flow table pipeline to switch hardware flow table pipeline as shown in Fig. 7. The same rules can be fitted into different types of hardware or switches by using the middle layer in FlowAdapter. It is define as an OpenFlow switch by combining a software switch and the available traditional hardware switches. The software switch is capable of handling SDN rules. The middle layer uses OpenFlow switch for fast forwarding. The main limitation of FlowAdapter is the limited processing ability because FlowAdapter realizes in the hardware switch.

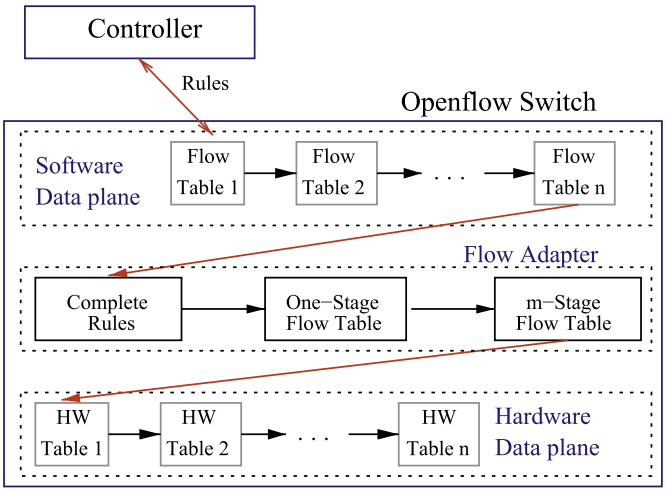


Fig. 7. FlowAdapter providing middle-ware for converting SDN flow rules to rules supported by traditional hardware (Pan et al., 2013).

An existing enterprise network consists large distributed devices and complex policies. The replacement of existing network with a centralized program for SDN is a challenging task. The centralized program usually demands designing of network policies from the scratch and unable to reuse existing traditional network policies. This issue of migration was addressed by Exodus (Nelson et al., 2015). The authors aimed to automatic migration of enterprise network configurations to SDN. Nelson et al. (2014) presented Flowlog, a tierless language for programming SDN controllers. Flowlog provided a unified abstraction for control plane, data plane, and controller state. It develops a unified program template module, that is equivalent to the existing network in terms of policies and operating features. It takes switch configuration as input and generates OpenFlow based table for each switch such that the system can mimic behavior of input configuration. The authors successfully implemented migration for dynamic routing, virtual local area networks, and network address translation.

4.1.2. Implementing SDN using hybrid network

The hybrid network consists the traditional devices and OpenFlow enabled devices as shown in Fig. 8. The hybrid network helps to reduce the network cost and provides the SDN functionalities (Guo et al., 2014; Hong et al., 2016; Sieber et al., 2016; Wang et al., 2017a; Jin et al., 2017; Xu et al., 2017a; Sinha et al., 2017; Wang and Huang, 2016).

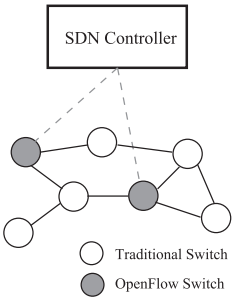


Fig. 8. Implementing SDN using hybrid network.

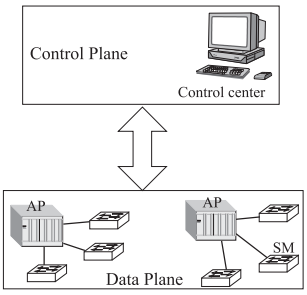


Fig. 9. Hybrid SDN based smart grid communication system.

The hybrid network can be implemented via an incremental method, *i.e.,* at a time certain traditional devices can be replaced with OpenFlow devices. The existing work focused on the following challenges upon creation of a hybrid network: identifying which nodes are to be replaced with the SDN nodes, obtaining global topology, and dynamic path update (Guo et al., 2014).

The authors in Wang and Huang (2016) used the concept of hybrid SDN to introduce a novel SDN based smart grid communication architecture. In smart grid communication, smart meters gather energy consumption and control information from residential and industrial consumers. As illustrated in Fig. 9, authors installed various aggregate points among the smart meter network such that the data generated by the smart meters can be forwarded to the control plane via the aggregate points.

The paper Guo et al. (2014) proposed an algorithm to deploy SDN switches with existing regular nodes (*e.g., switches, router*) in a network. SDN switches are controlled by the centralized controller. The open shortest path first routing protocol runs on SDN switches and regular node. This computes the global network topology. All the flows that accumulates at the SDN switches can be split to balance the flow in the network. The weights of network graph and flow division ratio of SDN switches need to be optimized for improving the system performance. The main limitation of this work is that it does not discuss which traditional node will be replaced with SDN switches. The authors in Hong et al. (2016) proposed two graph based heuristic approaches for solving such traditional nodes replacement problem. The authors in Jin et al. (2017) presented a unified network controller. It exerts fine-grained path control over SDN switches and regular switches in hybrid networks. The controller introduced magnet MAC addresses and dynamically updates MAC mappings at hosts. It is done with the help of gratuitous ARP messages for visibility and routing control. It used the ability of SDN switches to send “custom” packets into the data plane to manipulate regular switches. It update forwarding entries with MAC addresses for enhanced routing flexibility. The authors in Xu et al. (2017a) proposed depth-first-search and randomized rounding based algorithms. These algorithm used for throughput maximization routing in SDN hybrid network.

Due to flexibility of programming in SDN, it supports different network scenarios like metro or core network. In Sgambelluri et al. (2013b) authors used this flexibility for full control of coverage access and proposed a generalised hybrid controller for maintaining the quality of services.

4.2. Resource management

Consumable SDN resources include labels (*e.g.*, IP addresses, VLAN tags) and physical components (*e.g.*, switches, bandwidth, buffer space). Such resources need to be properly allocated and managed among various sub-networks, for proper operation and coordination among them. This section discusses the challenges and solutions in resource management of SDN, which is also summarized in Table 9.

4.2.1. Resource management in control plane

A distributed control plane shares the resources among different controllers in SDN. Each controller is responsible for managing a subnetwork in which switches can be dynamically connected to one or more controllers (Krishnamurthy et al., 2014). The policies defined by the applications can also be installed on any of the working controllers and are required to be effectively combined to obtain a global policy (Dixit et al., 2014b; Baldin et al., 2014; Hassas Yeganeh and Ganjali, 2012; Dai et al., 2014).

The authors in Dixit et al. (2014b) proposed a framework for creating a control plane using controllers from different vendors. The framework applied services, implemented on heterogeneous controllers to the same network traffic. The framework implemented as a layer between data plane devices and heterogeneous controllers using standardized protocols. It specified policies defining how services from different controllers are combined to modify the data path. A policy is described by a flow and an ordered set of controllers whose services should be applied to that flow. The framework modified the messages from the controllers according to the defined policy. A combined flow table pipeline is installed on the switch. The main limitation of this work is the bottleneck problem. It arises because all the northbound and southbound traffic flows through the framework.

Another resource management framework was proposed in Baldin et al. (2014). The framework included a central resource manager, domain resource managers, authorization framework and domain controllers. The central resource manager grants resources based on the resource request from the domain managers and available resources. Domain resource managers can allocate their resources to SDN controllers or resource managers of sub-virtual domains. The SDN requires hierarchical control for managing policies and rule conflicts. During the implementation of flow rule in a resource shared environment.

The authors in Ferguson et al. (2013) presented the design, implementation, and evaluation of an API for controlling the applications in SDN. The API implemented by an OpenFlow controller that delegates the authority to end users. The users provide requests in the form of a share tree as shown in Fig. 10. If the user is authenticate, the request is incorporated into the policy tree via Hierarchical Flow Tables (HFTs). So, the controller can install the policy by resolving all the conflicts.

Another graph theory approach was used in Prakash et al. (2015) that presented a framework known as Policy Graph Abstraction (PGA). The Policies in PGA are expressed by the operators in terms of intuitive and graph-based abstractions. Several different policy graphs are combined into a single graph that contains constraints for all the inputs. The authors used a syntactic policy composition operator as shown in Fig. 11. The composition of graphs in the PGA is done using graph unions and the conflicts which arise are notified to the network operator. The PGA supports automated and eager composition of modular policies.

4.2.2. Resource management in data plane

The resources in data plane include available Ternary Content Addressable Memory (TCAM) in hardware switch, CPU processing in software switch, and bandwidth in the connecting links. Various network applications (*e.g.,* flow accounting, load balancing, traffic engineering, and performance analysis) depend on the efficient measurement of the different type of network flows. Tasks such as traffic measurement require flow entries to be installed in SDN switches. The switches used in SDN uses TCAM and Dynamic Random-Access Memory (DRAM). TCAM is limited in size, power consuming, and costly. DRAM on the other hand is slow and managed by the CPU which is a resource constraint. Therefore the resources in SDN switches need to be managed efficiently while defining rules based on different policies (Moshref et al., 2013, 2015; Ferguson et al., 2012; Pereíni et al., 2014).

The problem of managing resources required by the measurement tasks is addressed in Moshref et al. (2015). The authors described

Table 9
Existing Challenges and solution of Resource Management in SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Das et al. (2017)	Inter-domain QoS negotiations	Branch and bound-based algorithm	–	–	Theoretical	–	RM in data plane
Basta et al. (2017)	Minimizing network cost and data resources	Optimal placement of data center and NFV	–	–	Simulated	Mobile Core topology	RM in data plane
Zhu et al. (2017)	Minimize anchors in WSN	Centralized anchor scheduling scheme	–	OpenFlow	Simulated	–	RM in data plane
Wang et al. (2018)	Control-plane communication	Flow redirection	Opendaylight	OpenFlow	Testbed	–	RM in control plane
Pedreno-Manresa et al. (2018)	Resource allocation in RAN	Resource orchestration techniques	–	–	Simulated	–	RM in data plane
Yang et al. (2017)	Joint admission control and routing	Approximate dynamic programming	POX	OpenFlow	Simulated	–	RM in control plane
Chen et al. (2016b)	Resource management via network	Dynamic resource management scheme	Opendaylight	OpenFlow	Testbed	–	RM in data plane
Blenk et al. (2016)	The optimal placement of the hypervisor	Mixed integer programming formulations	–	OpenFlow	Simulated	Abilene, Bellcanada, Dfn	RM in control plane
Wang et al. (2016b)	Virtual resource allocation problem	Discrete stochastic approximation		–	Simulated	–	RM in data plane
Mu et al. (2016)	User fairness in HTTP adaptive streaming	Hierarchical fair resource allocation model	Ryu	OpenFlow	Testbed	–	RM in data plane
Cai et al. (2016)	Imperfect network state information	Discrete stochastic approximation algorithms	–	–	Simulated	–	RM in data plane
Li et al. (2015a)	Optimization and unified control of NFs and forwarding devices	Heuristic approaches to ILP formulation	–	–	Simulated	Backbone, Fat-Trees, B4	RM in data plane
Maksymyuk et al. (2016)	Coexistence of a Wi-Fi network and an LTE	A resource management scheme	–	–	Simulated	–	RM in data plane
Cziva et al. (2016)	Reduce the network wide communication cost of DCs	A distributed, measurement based live VM migration algorithm	Ryu	OpenFlow	Simulated	Canonical tree	RM in control plane
Riggio et al. (2015)	Programming astractions for wireless networks	Software Defined Radio Access Network Controller for Enterprise	–	–	Testbed	–	RM in data plane
Tuncer et al. (2015)	Network resource management	A placement algorithm	–	OpenFlow	Simulated	Abilene, Geant	RM in data plane
Szabo et al. (2015)	Control of resources for elastic services	Unifying Compute and Network Virtualization	–	–	Theoretical	–	RM in data plane
Thyagaturu et al. (2016)	Resource allocation in backhaul network	SDN based resource allocation	–	–	Simulated	–	RM in data plane
Rückert et al. (2016)	Fine-granular traffic engineering of multicast	Network-layer multi-tree approach	Ryu	OpenFlow	Simulated	–	RM in data plane
Tan et al. (2014)	Control of converged networks	SDN based control architecture	–	OpenFlow	Simulated	–	RM in data plane
Dixit et al. (2014b)	Composing services from different vendors	Standardized controller to switch	Floodlight	OpenFlow	Simulated	–	RM in control plane
Baldin et al. (2014)	SDN resource delegation problem	Slice network into multiple resource dimensions	Floodlight	OpenFlow	Testbed	GENI	RM in control plane
Ferguson et al. (2013)	Decompose control and visibility of the network	Delegation of privileges and hierarchy of conflict	PANE	OpenFlow	Prototype	–	RM in control plane
Prakash et al. (2015)	Composition of independent network	Graphs composition, Graph union	POX	OpenFlow	Simulated	–	RM in control plane
Moshref et al. (2015)	Limited TCAM and accuracy of measurements	Dynamic resource allocation	Floodlight	OpenFlow	Prototype	CAIDA	RM in data plane
Moshref et al. (2013)	Resources trade-offs in measurements	Max cover Hierarchical Heavy Hitters	–	–	Simulated	CAIDA	RM in data plane
Ferguson et al. (2012)	Hierarchical policies in SDN	Hierarchical Flow Tables	PANE	OpenFlow	Simulated	–	RM in data plane
Pereíni et al. (2014)	Network state updates	Update installation as scheduling problem	POX	OpenFlow	Testbed	FatTree	RM in data plane

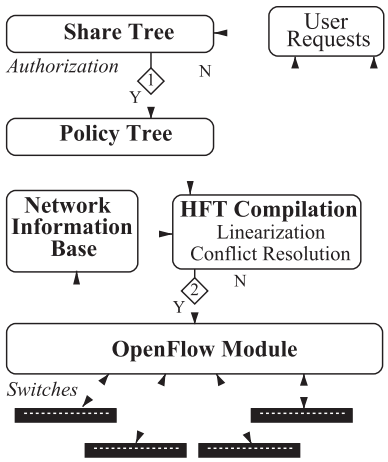


Fig. 10. Policy combining in resource shared SDN environment (Ferguson et al., 2013).

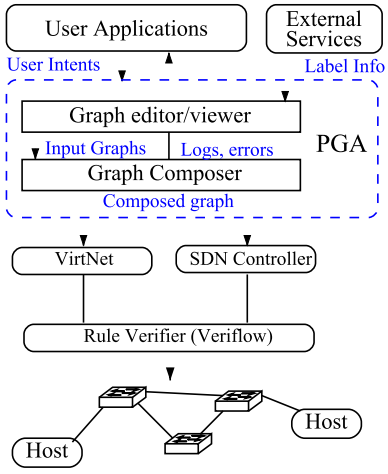


Fig. 11. Illustration of policy graph abstraction framework (Prakash et al., 2015).

an adaptive measurement framework, called DREAM. It dynamically adjusts the resources devoted to each measurement task, while ensuring a user-specified level of accuracy. The framework takes advantage of temporal and spatial multiplexing available in the data plane. Temporal multiplexing refers to a fact that the TCAM requires changes with time. Spatial multiplexing refers to the fact that different switches require different TCAMs. Software methods are also available for measurements along with the TCAM counters such as hash-based technique, and small code fragments in switches. These methods require different resources, hash-based data structures need SRAM, and code fragments require CPU for processing. Authors in Moshref et al. (2013) presented software-based measurement algorithm, which uses a combination of above measures based on resource availability and accuracy required. The detection of hierarchical heavy hitters (a measurement task) is demonstrated under various flow conditions. Hierarchical Flow Tables (HFTs), defined in Ferguson et al. (2012), uses a tree of policies and specify custom conflict-resolution operators at each node in the tree. Each node checks the packet headers and values then generates an action set. Based on the header values, the packet moves along the policy tree accumulating the required actions. A conflict resolution operator like child overrides parent and left the child in inter-sibling conflict, are used. A compiler and a runtime system to realize policy trees on a network of OpenFlow switches are also designed.

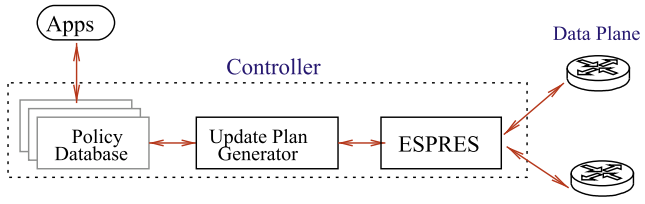
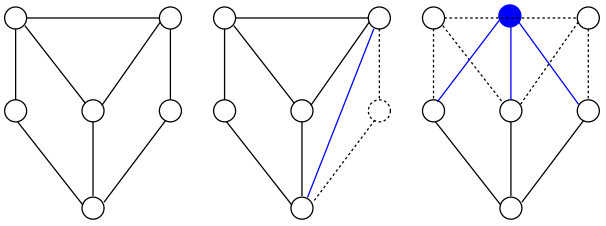


Fig. 12. Illustration of easy scheduling and prioritization for SDN (Pereíni et al., 2014).



(a) Physical topology (b) Link virtualization (c) Node virtualization

Fig. 13. Illustration of topology abstraction (Belbekkouche et al., 2012).

The authors in Pereíni et al. (2014) solved rule installation on constrained switches as a scheduling problem. Easy scheduling and prioritization for SDN uses queue manager that maintains command queues in the controller. Queue manager uses a scheduling algorithm to manage the order in which the commands are sent to the switches when switch queues are free. The controller architecture is sub-divided into layers, policy manager database, and an update plan generator as shown in Fig. 12. Any change required by applications are made by changing the network state in policy database. Update plan generator translates these changes into switch rules.

4.3. Virtualization

Virtual machines allow applications to flexibly run their programs without worrying about the specific details and characteristics of the underlying computing platform. Multiple virtual SDNs (vSDNs) can be defined over a physical network to share network resources (Belbekkouche et al., 2012). The topology of a network considers nodes (switches) and links between them. The operation in a large network depends on the abstraction or isolation of a part of network traffic from other. It can be achieved by using network virtualization. The network virtualization allows multiple isolated virtual networks to be built using the same physical network. Fig. 13 illustrates topology abstraction where part (a) of the figure shows the physical network. Parts (b) shows an example of link virtualization on the physical network where virtual link is depicted by blue color line. Part (c) shows an example of node virtualization where virtual node is shown by solid blue circle. Different levels of virtualization in SDN and work performed under them, are discussed in the rest of this section. We also present the summary of the existing work in the virtualization in network implementation of SDN in Table 10.

4.3.1. One-to-one mapping

The lowest level of topology virtualization is one in which the physical and virtual topologies are in one-to-one mapping. At this level, only the resources of nodes or links are shared among different subnetworks. Resources of a node mainly include the CPU resources and the flow table resources, whereas the shared link resource usually refers to link bandwidth.

Table 10
Existing challenges and solution of Virtualization in Network Implementation of SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Aujla et al. (2018a)	Inter DC migration overhead	Energy-aware flow-scheduling scheme	–	OpenFlow	Simulated	–	One to many mapping
Song et al. (2017)	Control path reliability problems	Control path reliability management framework	Floodlight, ONOS	OpenFlow	Simulated	–	All to one mapping
Yousaf et al. (2017)	Advent of 5G networks	SDN and NFV	–	–	Theoretical	–	One to many mapping
Yang et al. (2017)	Joint admission control and routing	Approximate dynamic programming	POX	OpenFlow	Simulated	–	All to one mapping
Chen et al. (2016b)	Resource management via network virtualization	Dynamic resource management scheme via slices	Opendaylight	OpenFlow	Testbed	–	One to many mapping
Blenk et al. (2016)	The optimal placement of the hypervisor instances	Mixed integer programming formulations		OpenFlow	Simulated	Abilene, Dfn, Quest, OS3E	One to many mapping
Wang et al. (2016b)	Virtual resource allocation problem	Discrete stochastic approximation	–	–	Simulated	–	One to many mapping
Faraci and Schembra (2015)	Orchestration and management of SDN/NFV	A green management policy		OpenFlow	Simulated	–	One to many mapping
Riggio et al. (2015)	Programming astractions for wireless networks	Software Defined Radio Access Network Controller for Enterprise	–	–	Testbed	–	One to many mapping
Bouten et al. (2017)	Affinity-constrained SFC placement problem	A semantic conflict detection mechanism	–	–	Simulated	BRITE	One to many mapping
Chen et al. (2016c)	Network virtualization for layer 2 networks	SDN based network architecture	Floodlight	OpenFlow	Testbed	Fat-Trees	One to many mapping
Kang et al. (2013)	Optimizing placement of rules	Divide and conquer	–	–	Theoretical		All to one mapping
Soulé et al. (2014)	Resource management	Global network policies	–	OpenFlow	Testbed	Topology Zoo	All to one mapping
Gutz et al. (2012)	Achieving isolation in networks	Slice abstraction	–	–	Theoretical		One to many mapping
Lin et al. (2012)	Resource virtualization	Virtualization cloud platform in SDN	–	OpenFlow	Simulated	–	One to many mapping
Al-Shabibi et al. (2014)	Infrastructure as a service for SDN	Network hypervisor	Floodlight	OpenFlow	Simulated	Internet2	One to many mapping
Ghorbani and Godfrey (2014)	Behavior differences in virtual to physical mappings	End to end correctness for virtual physical mapping	–	OpenFlow	Theoretical	–	One to many mapping
Syed and Van der Merwe (2016)	Mobile network service management	Services as templates	–	OpenFlow	Simulated	PhantomNet	One to many mapping
Velasco et al. (2018)	Proactive and reactive network slicing	Architecture of automatic network slicing	ONOS	OpenFlow	Simulated	Net2Plan	One to many mapping

Table 11
Network performance measures in SDN.

Network Performance Measures	Latency	Programmability of Data Plane (Chen et al., 2015; Borokhovich et al., 2014; Song, 2013; Moshref et al., 2014; Mekky et al., 2014; Saunders et al., 2016; Lu et al., 2012; Newport and Zhou, 2015) Graph Theory Approach (Ros and Ruiz, 2014; Jin et al., 2016; Krishnamurthy et al., 2014; Heller et al., 2012; Wang et al., 2016c)
	Efficiency	Routing Efficiency (Vissicchio et al., 2015; Chen et al., 2016d; Hartert et al., 2015; Huang et al., 2016a; Sgambelluri et al., 2016; Bidkar et al., 2014; Hao et al., 2016; Martini et al., 2015; Guo et al., 2016; Paolucci, 2018) Efficient Memory Utilization (Nguyen et al., 2014; Yan et al., 2014; Katta et al., 2013; Rifai et al., 2015)
	Consistency	Network State Consistency (Vanbever et al., 2013; Ghorbani and Caesar, 2012) Rule Update Consistency (Katta et al., 2013; McGeer, 2012)
	Traffic Measurement	Measurement using Traffic Differentiation (Gibb et al., 2012; Yu et al., 2014; Gong et al., 2015; Sun et al., 2015a) Measurement using Additional Tool (Agarwal et al., 2014; Dwaraki et al., 2015).

4.3.2. All-to-one mapping

At the highest level of virtualization, the entire physical network is abstracted as a single virtual node. The biggest challenge in implementing the entire network as a single virtual switch abstraction is mapping global, high-level policies to individual switch rule in the network. The authors in Kang et al. (2013) proposed a mapping scheme and presented a rule-placement algorithm. It takes global network policy as input and generates the switch level optimized flow rules. The paper (Soulé et al., 2014) presented a network programming language called merlin language. This language define global network policies and a compiler to convert these policies to switch rules.

4.3.3. One-to-many mapping

Most of the virtual networks employ one-to-many mapping (Lin et al., 2012; Al-Shabibi et al., 2014; Ghorbani and Godfrey, 2014; Gutz et al., 2012; Monsanto et al., 2013) in which a node or link is mapped to multiple node or link as shown in Fig. 13. Authors in Gutz et al. (2012) shown that the isolation feature can be achieved by the programming language for managing different virtual networks. The network is divided into slices and each slice is programmed independently. It will allow the compiler to deal with the implementation of isolations, by combining and managing separate programs. It will reduce the load of maintaining virtual networks or placement of middle-boxes. A Virtualization Cloud Platform (VCP) is proposed in Lin et al. (2012) that located upon the network operating system as the running environment for applications. VCP provides standardized Network Application Programming Interface (NetAPI) to various applications in SDN. VCP provides resource virtualization, which includes topological abstraction and bandwidth virtualization. It however requires a large number of databases for storing APP-IDs, physical network information, virtual network information, and mapping information. The authors in Al-Shabibi et al. (2014) proposed a similar approach, known as OpenVirtex (OVX), that enables different network operators to create and manage virtual SDN. OVX provides address virtualization to manage separate traffic of all networks and topology virtualization to allow operators to specify their topology requirement. Each tenant is also provided with its own network operating system for managing its own network.

The authors in Ghorbani and Godfrey (2014) shown that many-to-one mapping is mainly a means to share resources. One-to-many mapping provides basic support for functional mobility, enables a scale-out approach to network design. In this network design additional physical networking elements scale-out a single logical abstraction, provides high-availability, and overcomes lack of capacity of physical elements. The authors answered the following question: *could the one-to-many mapping lead to incorrect behavior in the network?*. The authors demonstrated the unusual behaviors that could result from a one-to-many mapping. The paper (Syed and Van der Merwe, 2016) presented Proteus, a mobile network service control platform to enable safe and rapid evolution of services in a mobile software defined infrastructure. Proteus allows for network service and network component functionality to be specified in templates.

5. Network performance

Network performance is a metric which indicates the quality of service of the network. An SDN includes the following perform matrices: network latency, packet forwarding, efficiency, and flow rate. This section will elaborate all the performance matrices of the SDN as presented in Table 11.

5.1. Latency

Network latency is the amount of time a packet takes to travel from one designated point to another. It decides the performance of any network dominantly. Acceptance and rejection of any developed model is decided by its latency. Initial packets of a new flow in SDN forwarded to the controller for computing the flow rules. Thus, the estimation of latency in SDN also adds the forwarding latency of the initial packets. In dynamic network, large number of unmatched packets arrive at the switches for which the forwarding decision is made by the controller. Such forwarding overhead between the switches and controller for every decision, increases the latency of the network. In the case of large networks with a single controller, the time taken by switches to communicate with the controller is high. The latency of a network can be analyzed using various metrics such as average and worst case latency. Studies focusing on latency can be categorized based on approaches of solutions, *e.g.,* programmability of the data plane and graph based approach. Table 12 summarizes the existing challenges and solution of Latency in SDN.

5.1.1. Programmability of data plane

A switch in SDN forwards the packets of a new flow to the controller for computing the flow rules and receives the rules from the controller. Such request and reply messages for rules finding in SDN increases the latency of the network. Increasing the programmability of the switches in SDN can help to reduce the rules finding overhead and therefore improves the latency of the network. The authors in Song (2013), Moshref et al. (2014), Mekky et al. (2014), Borokhovich et al. (2014), Chen et al. (2015), Saunders et al. (2016), Lu et al. (2012) and Newport and Zhou (2015) proposed various methods for improving the latency of the network.

The SDN switches support the following two design methods. The first method uses single on-chip ASIC for performing all operations (*e.g.,* packet parsing, lookup, buffering, scheduling and modification). The lookup tables and packet buffer are also embedded in the ASICs. A CPU

Table 12
Existing challenges and solution of Latency in SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Li et al. (2018)	QoS guarantee in satellite communication	QoS-oriented heuristic algorithms for routing	POX	OpenFlow	Prototype	Tr Constellation	Latency
Zhao and Medhi (2017)	Lack of network awareness in M/R framework	Using SDN architecture for Hadoop	Ryu	–	Testbed	GENI	–
Liu et al. (2018)	Optimal use of limited wireless fronthaul	Lyapunov stochastic optimization	–	–	Simulated	–	Latency
Wang et al. (2017b)	Control link load balancing and low delay route deployment problems	Minimum-cost flow redirecting (MCFR) algorithm	–	–	Simulated	Fat-Trees, Monash University	Programmable data plane
Gharakheili et al. (2016)	Bandwidth bottleneck in access networks	Two-Sided Resource Management	Floodlight	OpenFlow	Simulated	–	Programmable data plane
Lin et al. (2018)	Overhead of self healing in PMU network	Greedy heuristic algorithm to ILP model	–	–	Testbed	–	Programmable data plane
Liu et al. (2016b)	Problem of cooperative data scheduling	Greedy method	–	–	Simulated	–	Programmable data plane
Varvello et al. (2016)	Packet Classification in SDN	GPU based packet classification	–	–	Simulated	–	Programmable data plane
Cello et al. (2016)	Predict packet loss in SDN	Neural Predictor	–	–	Simulated	–	Programmable data plane
Sood et al. (2016)	Study performance of SDN switches	Queuing theory-based model	–	–	Simulated	–	Programmable data plane
Wang et al. (2016d)	Joint-bandwidth allocation problem	Constrained utility-optimization formulation	NS-2	–	Simulated	G-WAN, Softlayer	Programmable data plane
Wu et al. (2015)	Scalability of visual data transmission in RAN	Integrate SDN and C-RAN		–	Simulated	–	Efficiency
Xue et al. (2015)	Multi-source SVC video problem	Heuristics to ILP formulation	POX	OpenFlow	Testbed	–	Programmable data plane
Martinello et al. (2014)	Table lookup in the data plane	Stateless routing/forwarding concept	–	OpenFlow	Simulated	–	Programmable data plane
Li et al. (2016b)	High latency in VANET communication	An optimal rebating strategy	–	–	Simulated	–	Programmable data plane
Heller et al. (2012)	Controller placement problem	Optimal controller placement using various metrics	–	–	Simulated	Internet2	Graph theory approach
Jin et al. (2016)	Switch complexities	Source routing	–	–	Simulated	–	Graph theory approach
Wen et al. (2014)	Rule update latency	Extended policy compiler		OpenFlow	Simulated	–	Graph theory approach
Lu et al. (2012)	Programmability of data plane	Adding CPU to switch	–	–	Prototype	–	Programmable data plane
Mekky et al. (2014)	switch controller delay	Adding application logic to switch	Floodlight	OpenFlow	Simulated	–	Programmable data plane
Moshref et al. (2014)	Controllers involvement in new flows	State transition logic in switch	POX	OpenFlow	Simulated	–	Programmable data plane
Song (2013)	Limited programmability of openflow	Assembly language for data plane	Floodlight		Simulated	–	Programmable data plane
Newport and Zhou (2015)	Switch controller delay	In network decisions in data plane	–	–	Theoretical	–	Programmable data plane
Paris et al. (2016)	Flow reconfigurations in SDN	Control policy for reconfigurations using ILP	–	–	Simulated	GEANT	Graph theory approach

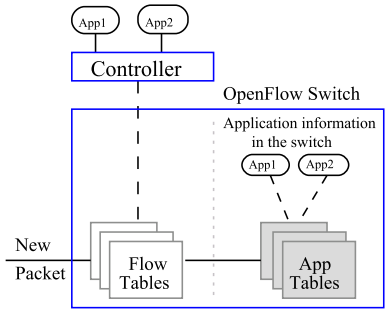


Fig. 14. An application-aware SDN architecture (Mekky et al., 2014).

and DRAM are embedded with the switch in the second method for handling complex functions (Song, 2013). The single on-chip ASIC method simplifies the data plane but the designed switches have limited storage space for flow tables and on-chip buffer. On the other hand processing power of the CPU is increasing which shifts the design trend towards utilizing the CPU within switches (Lu et al., 2012). Presence of CPU and DRAM embedded with switch provides the following advantages, complete forwarding tables can be moved to software tables reducing the space required for ASICs, large traffic flow handling can be offloaded to the CPU, and as DRAM has a high storage capacity it can be used to buffer large traffic flows.

The CPU can also be programmed to analyze the traffic and identify the mitigate malicious traffic. All these methods require modifications to be made within the switches and therefore forwarding in DRAM is slow. Authors in Newport and Zhou (2015) claimed that by using only the SDN data plane switches and their available standard rules, various network management tasks can be solved without involving the controller. It was theoretically proved that the SDN data plane, using only the available forwarding rules, can solve any problem that can be solved by a Turing Machine in polynomial time.

The matching fields in data plane are limited till layer four protocols. The switches are designed to be stateless, *i.e.,* take action on individual packet not based on the saved flow. Therefore, the applications like NAT and context aware routing, cannot be handled in the data plane. In paper (Mekky et al., 2014), an application-aware SDN architecture is proposed that enables the data plane to handle applications requiring Layer 4 to Layer 7 and stateful protocols. This is achieved by keeping application logic locally on the switches for taking decisions as shown in Fig. 14. Flow-level State Transitions (FAST) (Moshref et al., 2014) used state machines for handling dynamic applications. The network operators define the state machines which are then installed on the switches. Open vSwitches are used as switch agents that manage a pipeline of tables to support state machines with the commodity switch component. When a hardware switch encounters a flow for which no rule exists in the flow table. The hardware switch sent the flow to the switch agent which processes it through the state machine pipeline. Observed drawback included delay in state machine installation on all switches. Also the addition of new applications requires states machines to be added reactively. It provides a good programing ability but it requires modification to be made in existing OpenFlow switches which is hard to accomplish.

5.1.2. Graph theory approach

The performance of an SDN depends highly on the topology, *i.e.,* placement and number of switches and controllers used in the network design. Graph theory based approaches can be used for finding efficient solutions to different problems (*e.g.,* controller placement, topology discovery, link assignment, load balancing). The existing work proposed in Ros and Ruiz (2014), Heller et al. (2012), Jin et al. (2016) and Krishnamurthy et al. (2014) reduces network latency and improves network performance. Optimal deployment of controllers can be obtained by computing latency performance metrics for all possible deployments (Heller et al., 2012). Results inferred from various simulations conclude that latency in the network decreases almost proportionally with increasing number of controllers. The random placement of controllers in the network does not provide optimal results. Authors in Wang et al. (2016c) proposed an approach for assigning switches to the controllers dynamically using the principle of the game theory. First, a stable matching between the controllers and the switches is found. Switch prefers a controller with low response time to reduce latency. A controller prefers to manage those switches which are topologically closer so as to reduce the control traffic overhead. Once a matching is obtained the coalition game phase algorithm is used to dynamically balance the matching based on network load. A different approach was adopted in paper (Jin et al., 2016) in which an architecture called Sourcey was defined. It applies the concepts of source-based routing and labels for forwarding packets. The controller computes the path between source and destination, defined as a path stack and stores it in a packet header. Each switch, when it receives a packet, pops the top label that specifies the output port of the packet. The Sourcey eliminates the need of individual switch configuration and therefore decreases latency and increasing flexibility.

Dynamic changes in the network topology force the controller to periodically update forwarding policies. Reconfiguring rules in flow tables creates, overhead as the rules are to be synchronized throughout the network. Policies are compiled with high-level compilers that do not consider the rule dependencies. It generate redundant rules in update procedure. The authors in Wen et al. (2014) shown that the network latency can be minimized by eliminating unnecessary updates. This can be achieved by mapping rule dependency and priority levels in a Directed Acyclic Graph (DAG) while generating updates. The DAG is created by the compiler while generating the policy rules. Another method for minimizing flow rule reconfiguration is proposed in Paris et al. (2016). The authors assumed a limit on reconfiguration of the network topology based on available network budget. To limit the reconfigurations, the authors proposed two control policies that minimize the flow allocation cost within a network reconfiguration budget. The authors also validated the model by experimenting with a realistic network setting and using standard Linear Programming tools used in the SDN industry.

5.2. Efficiency

Efficiency parameter measures *how efficiently the information is exchanged within the network?*. It mainly deals with forwarding rule storage and management, packet processing and forwarding in the network, and coordination between software and hardware switches to achieve high performance. Efficiency measures of SDN, its existing challenges and solution are presented in Table 13.

5.2.1. Routing efficiency

Designing an efficient routing mechanism for forwarding the packets from the switches to the controller, is a challenge in the SDN. Routing mechanism should be such that there is an efficient utilization of the network resources with no links or nodes to be congested. Therefore, there is a need of a routing scheme that incorporates constraints of link, bandwidth, and the available forwarding table size (Chen et al., 2016d; Hartert et al., 2015; Vissicchio et al., 2015; Huang et al., 2016a). SDN has many advantages to support efficient routing due to its characteristics such as isolation of control and forwarding plane, global centralized control, and programmability of the network.

The flows in SDN compete with each other and fairly share the link bandwidths with each other. Such existing scheduling technique is called fair share routing, which uses the TCP congestion control algorithm. However, fair or homogeneous sharing of the link bandwidths reduces the utilization of resources and thus wasting the energy.

Table 13
Efficiency measures of SDN, its existing challenges and solution.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Wang et al. (2017c)	Limited size of flow tables in switches	Multicast Addressing and header modification	Floodlight	OpenFlow	Simulated	–	TCAM utilization
Soni et al. (2017)	Problems with multicast services in SDN	Threshold-based loadbalancing algorithm	–	OpenFlow	Simulated	INTERNET2-AL2S	Routing
Pan et al. (2018)	Action composition problem	Graph based approach	–	–	Simulated	–	TCAM utilization
Coronado et al. (2018)	Multicast over WiFi	Joint mobility management	–	–	Testbed	–	Routing
Wang et al. (2017d)	Study of computation diversity	Combining aspects of SDN, CRAN	MCC	–	–	Theoretical	Efficiency
Xu et al. (2018)	Scalability challenge in SDN	Hybrid switching (HS) design	–	OpenFlow	Testbed	HyperX	TCAM utilization
Wang et al. (2017e)	VM migration in SDN	Fully polynomial time approximation (FPTA)	–	–	Simulated	Topology of PRV1, B4	Reduce migration time
Wu et al. (2017)	Bulk transfers in data centres	Job Scheduling approach	Beacon	OpenFlow	Prototype	–	Chunk Routing
Bao et al. (2018)	Traffic imbalance due to Node mobility	Supervised learning for prediction	NS-3	–	Simulated	–	Routing
Fernández-Fernández et al. (2017)	QoS requirements and energy awareness	Multi-objective evolutionary algorithm	–	–	Simulated	Abilene	Routing
Zheng et al. (2017)	Inconsistencies in network updates	Congestion- and loop-free network updates	Dpctl	OpenFlow	Simulated	–	TCAM utilization
Yang et al. (2017)	Joint admission control and routing in video streaming service	Approximate dynamic programming	POX	OpenFlow	Simulated	–	Routing
Huang et al. (2016b)	Packet delay due to caching rules in switches	A rule partition and allocation algorithm	–	–	Simulated	–	TCAM utilization
Guck et al. (2016)	Guarantee hard real-time QoS in SDN	Function Split Between Routing and Resource Allocation	–	–	Testbed	Ring	Routing
Huang et al. (2016c)	Limited TCAM memory and rule caching	Approximation algorithm	–	–	Simulated	–	TCAM utilization
Xing et al. (2016)	Detecting large flows in SDN	Sampling and flow counting for flow measurements	–	OpenFlow	Simulated	CAIDA	TCAM utilization
Sun et al. (2018)	Lookup bottleneck in distributed file systems	SDN based metadata lookup service	–	OpenFlow	Testbed	–	TCAM utilization
Polverini et al. (2016)	Traffic matrix estimation	Flow spread-based algorithm		–	Simulated	Nobel, Geant, Germany	TCAM utilization
Li et al. (2015b)	Inefficient network energy usage of DC	Exclusive routing for each flow	–	OpenFlow	Testbed	Fat-Trees	Routing
Huang et al. (2015)	Rule placement problem	Rule multiplexing scheme	–	–	Simulated	ITALYNET	TCAM utilization scheme
Amokrane et al. (2015)	Energy consumption of wireless access networks	Online flow-based routing approach	–	–	Simulated	–	Routing
Martinello et al. (2014)	Table lookup in the data plane	Stateless routing/forwarding concept	–	OpenFlow	Simulated	–	TCAM utilization
Dong et al. (2015)	Limited size of flow tables in switches	A rule caching mechanism	Opendaylight	OpenFlow	Simulated	–	TCAM utilization
Lai et al. (2015)	Selection policy in 5G networks	A buffer-aware HTTP live streaming approach	–	–	Simulated	–	Routing
Yan et al. (2014)	Rule dependencies	Wildcard rule caching	–	OpenFlow	Simulated	–	TCAM utilization
Nguyen et al. (2014)	Number of rules in SDN	Flow rule optimization	–	OpenFlow	Simulated	–	TCAM utilization
Li et al. (2014c)	Power consumptions in data centres	Graph based approach	–	OpenFlow	Simulated	Fat-Tree	Routing
Hartert et al. (2015)	Network optimizations	Constraint programming	–	–	Simulated	Rocketfuel	Routing
Vissicchio et al. (2015)	Centralized routing decisions	Divide and conquer	–	–	Prototype	–	Routing
Huang et al. (2016a)	Quality service to users	Graph based approach	–	–	Simulated	–	Routing
Cohen et al. (2014)	Limited size of flow tables in switches	Software switch	–	–	Simulated	B cube, mesh, BA	TCAM utilization
Paolucci (2018)	Multi layer network segmentation	Control architecture for segmentation	Ryu	OpenFlow	Simulated	–	Routing
Guo et al. (2016)	Throughput maximization in SDN	Online and offline routing algorithm	–	OpenFlow	Simulated	GEANT, Fat-Tree, BA	Routing
Martini et al. (2015)	Dynamic service chaining	Context aware delivery of application	SO-SDN	OpenFlow	Simulated	–	Routing
Hao et al. (2016)	Segment routing based recovery	Primal dual algorithm for restoration	–	–	–	Random	Routing
Bidkar et al. (2014)	SDN for service provider	Framework for service provider network	–	OpenFlow	Prototype	–	Routing
Sgambelluri et al. (2016)	Segment Routing	Enhance OpenFlow controller	SR-Controller	OpenFlow	Simulated	–	Routing

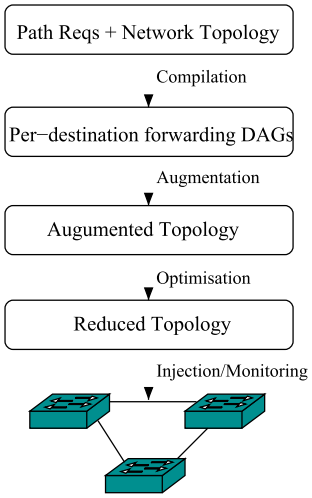


Fig. 15. Illustration of a new architecture in Vissicchio et al. (2015).

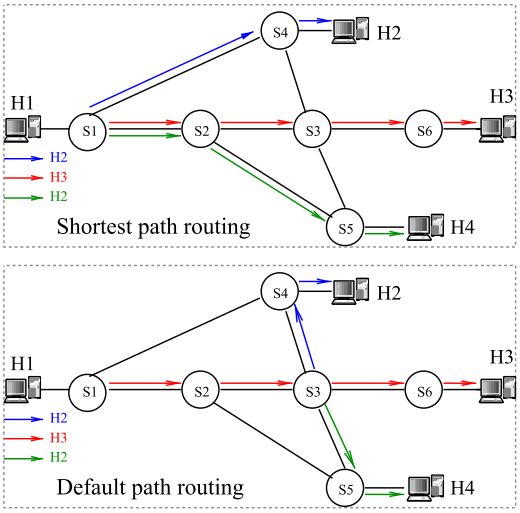


Fig. 16. Illustration of an optimization problem for reducing the TCAM storage requirement (Nguyen et al., 2014).

Fair-share of resources leads to idle state of controllers. This can be explained by the fact that controller will not get sufficient flow control request from all the switches. The exclusive routing proposed in Li et al. (2014c), aims to reduce the overall energy consumptions in the data center. The exclusive routing guarantees that every link is occupied by a single flow at a time. So, there are no bottlenecks in active links in the network. Each flow in the routing assignees some priority based on which the flow is scheduled. Such priorities depend upon the flow size, deadline, etc. The aim of this work is to maximize the number of flows that can be completed before the deadline.

The paper (Vissicchio et al., 2015) described an architecture, called Fibbing, that uses centralized control over distributed routing. The proposed architecture is expressive and flexible towards load balancing, link failures, and traffic engineering. The novel idea in this paper is that it introduces fake nodes and fake links in the network that allows forwarding any flows in any path based on dynamic requirement. The routing decisions are then taken on the abstract network topology computed using flow as shown Fig. 15. Another expressive architecture (Hartert et al., 2015) used middle-point routing for path selection and segment routing (Filsfils et al., 2015). It uses optimization layer for computing optimized path and avoid the limitations of the shortest path based routing.

In Paolucci (2018), authors proposed an architecture for changing service along the traffic path with the help of segment routing and traffic management. A path computation element handles the computation and placement of the controller whereas, segment routing allows automatic service enforcement. The above solution is highly useful in an elastic optical network. Authors in Guo et al. (2016) proposed a traffic merging algorithm in case of offline traffic routing. For online traffic routing authors proposed a primal dual update algorithm for meeting the quality of services. Service oriented SDN controller is presented in Martini et al. (2015) for addressing the challenges of dynamic service chaining in overlay networks. The main aim of authors was to provide context aware application service data transfer. Using SDN controller author's provision that data delivery path should be programmable for dynamic path establishment.

In case of failure recovery in a segmented network, determining a centralized primary path is essential for retrieving bandwidth. The authors in Hao et al. (2016) developed a randomized routing scheme for segmented routing for faster restoration. The SDN technology is getting its popularity in the service provider network. In Bidkar et al. (2014) authors presented a framework for supporting service provider network. This framework uses segmented routing through a software switch called software-defined ethernet switch routers. Authors in Sgambelluri et al. (2016) implemented segmented routing algorithms. The first implementation is done to obtain an enhanced version of an Openflow controller with data plane nodes having OpenFlow switch. The second implementation is for a novel path computation element. The data plane nodes in the novel path computation elements are IP/MPLS router. The authors further proved that both these implementations can be applied for rerouting of dynamic traffic.

5.2.2. *Efficient memory utilization*

SDN controller generates forwarding rules based on the network services. It installs them on the switches that perform forwarding of packet using the generated rules. Switches however, have limited memory to accommodate these rules. In general, forwarding rules are generated based on the efficient routing policies that optimize path length and latency. These rules are stored in the TCAMs that are expensive and have limited storage capability. Thus the rule generation and storage must be optimized for efficient TCAM utilization (Nguyen et al., 2014; Yan et al., 2014; Katta et al., 2013; Rifai et al., 2015). Forwarding rules stored in a switch can be exact rules with matching header fields or wild-card rules. Using wild-card rules improve the re-usability of rules in the flow table. As all flows related to a wild-card rule does not generate controller message. The number of packets forwarded to controller reduces.

The authors in Nguyen et al. (2014) proposed an optimization problem that gets the maximum amount of traffic delivered according to the policies and the actual dimension of the network. This shows that the minimum rules which can fulfill the policy requirement (*e.g.*, packets reach the destination irrespective of the path they travel) must be generated for improving TCAM storage efficiency. Number of rules on each switch were reduced by installing each node with a default forwarding rule and few rules that direct packets to the destination as shown in Fig. 16. Most of the packets arriving at a node (switch S1 and S2 as shown in the figure) can be forwarded by the default rule. It is done until they match a rule that diverts them to a path leading to the destination. A possible drawback of the model is increasing packet latency because of arbitrary paths used for forwarding the packet. It involves controller for forwarding packet that could not be directed to their destination point.

Source routing (Jyothi et al., 2015; Stephens et al., 2016) and label based routing (Chen et al., 2016a; Fang et al., 2015; Sinha et al., 2017; Jin et al., 2016) also generate less number of rules as compared to basic IP based routing schemes. The bounded path-degree max flow problem is investigated in Cohen et al. (2014). Here the goal is to maximize the overall feasible traffic that can be routed through the network. It

Table 14
Existing challenges and solutions of Consistency in SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Yaghoubi et al. (2018)	Inconsistency in network state due to reconfiguration	DP based control policy	–	–	Simulated	Backhaul	Network State consistency
Zhou et al. (2017)	Transient anomalies during asynchronous network updates	Lossless inter-domain route updates	–	OpenFlow	Simulated	–	Rule update consistency
Perešíni et al. (2018)	Diagnosing network anomalies	Boolean satisfiability formulation	–	OpenFlow	Testbed	–	Network State consistency
Zheng et al. (2017)	Inconsistencies in network updates	Congestion- and loop-free network updates via greedy method	Dpctl	OpenFlow	Simulated	–	Network State consistency
Salsano et al. (2016)	Integrating SDN in IP networks	Open Source Hybrid IP/SDN networking	–	–	Testbed	–	Additional tool
Mizrahi and Moses (2016)	Consistency in network updates	Accurate time based network updates	CPqD Dpctl	OpenFlow	Prototype	–	Network State consistency
Mizrahi et al. (2016)	Consistent network updates in SDN	Time-triggered network updates	–	OpenFlow	Testbed	Sprint, NetRail	Network State consistency
Levin et al. (2012)	Impact of distributed SDN on logical centralized application	Characterizing the state exchange points	–	–	Simulated	–	Network State consistency
Vanbever et al. (2013)	Upgrades in control plane	Hypervisor between controller and switches	Floodlight	OpenFlow	Simulated	–	Network State consistency
Ghorbani and Caesar (2012)	VM migration	Ordered planning using heuristics	–	–	Simulated	Random graphs	Network State consistency
McGeer (2012)	Consistent rule update	Boolean functions	–	–	Theoretical	–	Rule update consistency
Hua et al. (2016)	Consistency in network updates in adversarial setting	Flow ordered update	–	–	Simulated	–	Rule update consistency
Brandt et al. (2016)	Consistent migrations of flows	Polynomail time solution for splittable flows	–	–	Theoretical	–	Rule update consistency
Katta et al. (2013)	Network updates	Reachability analysis and mixed integer program	NOX	OpenFlow	Simulated	Fattree	Rule update consistency

must satisfy the limited bandwidth capacities of the links as well as the path-degrees or TCAM space of the switches. The problem is presented as an integral linear programming framework for which a bi-criteria approximation algorithm is proposed in the paper. The authors in Yan et al. (2014) proposed a wild-card rule caching system, called Caching in Buckets (CAB). It successfully provides reactive rules without creating any discrepancies in the network. CAB divides the rule set (or the address space) into small logical buckets. Rules are then associated with buckets according to their location in the address space. Caching is then performed for storing rules of a bucket together.

The software switches provide various advantages in SDN such as virtualization, middle-boxes, network flow virtualization, and flexibility in path selection (Lu et al., 2012; Song, 2013). For proper utilization of software switches in SDN, they must have a programmable data plane, high throughput, high packet rates, efficient CPU usage, and high port density. The available models fail to integrate all features within a single switch. A mSwitch (Honda et al., 2015) can fulfill all the above requirements. The authors extended virtual local ethernet switch model as it is interrupt based and provides high throughput. The mSwitch uses the switching fabric and split data plane approach for obtaining flexibility. A loadable kernel module is also used for switching logic. NetMap API is used for connecting virtual ports to VMs or applications. As NetMap API uses shared memory for isolation between applications. Each port has separate address space. As future work, mSwitch must include modules to receive entire batches of packets instead of having per-packet semantics to improve performance.

5.3. Consistency

The use of centralized controllers in SDN reduces the problem of network inconsistency. Large growing networks, however, cannot be controlled by a single controller. The presence of distributed controllers in the network suffers from the problem of inconsistency. The network state and forwarding tables are required to be efficiently synchronized, for the complete network consistency and proper functioning. Creating a strongly consistent logic requires a huge overhead and increases the latency of the system. If responsiveness has considered in the design, then the system becomes highly inconsistent. Two trade-offs in an SDN design have been observed in Levin et al. (2012). The first is control plane performance and system overhead and the second is logic complexity and inconsistency in the system. Literature related to existing challenges and solutions of consistency in SDN are given in Table 14. The existing work mainly consider the following two types of consistencies in an SDN.

5.3.1. Network state consistency

A high-performance system needs network state to be frequently synchronized among the controllers which helps to maintain a consistent global view. In the absence of the synchronization in the states, the network generates some errors such as the routing loop problem which may affect the network performance (Levin et al., 2012). Another measure to attain consistency is designing an application that is aware of the underlying system. It can compare the inter-domain view among the controllers.

The authors in Vanbever et al. (2013) presented a system, known as HOTSWAP, for upgrading SDN controllers in a disruption-free and correct manner. The system stores the required network state so that update occurs without any delay or errors. The working of the proposed HotSwap system is explained in Fig. 17. The system allows control transfer among the controllers with no packet loss and inconsistencies. The HotSwap can be used for achieving state synchronization in distributed controllers. Inconsistency in network state can also arise during virtual machine migration. The frequency of migration due to the lack of adequate network access and limited computing resources,

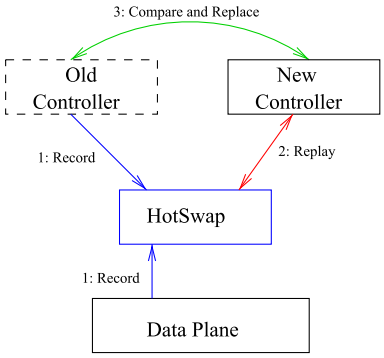


Fig. 17. Illustration of HOTSWAP system (Vanbever et al., 2013).

can disrupt the migration process. Given a start and end network configuration states along with the rules installed in the switches at both instances. An algorithm is proposed in Ghorbani and Caesar (2012) that finds out a sequence of instructions for making a seamless migration between states. The problem is solved in two stages. First, is obtaining a sequence of VM migration required and the second is finding instructions needed for each migration.

5.3.2. Rule update consistency

The modification, upgradation, and replacement are the routine activities in a software based programming system. An SDN also requires, above activities in a given period of time. The existing work that focuses mainly on the SDN maintain the desired performance of the system. A rule update is called consistent when during the update procedure either all packets throughout the network are processed by the old rule or all are processed by the new rule (McGeer, 2012; Katta et al., 2013). Achieving consistent updates in such a large network is a challenging task because of the distributed nature of the switches and controllers. It is very tough to predict the new policies that will arrive on the switches. When a packet arrives at the switch during a transition of state, it is unaware of the fact that other switches will handle the packet by which rule set (McGeer, 2012). It is thus not safe to forward the packet to the destination. The switches simply forward such packets to the controller, which can directly forward it to the destination. Limitation of McGeer (2012) is the assumptions that only two rule sets are present at a time. The protocol requires a large number of packets to be sent to the controller during transition. It causes an increase in network traffic and a burden on the control plane. This is due to installation of distributed set of proxies to handle the packets in SDN.

The authors in Katta et al. (2013) proposed an incremental update procedure that divides the complete update in the network into k non-overlapping stages. Each round selects a subset of packets that are updated from old policy to a new policy. Selection is done such that maximum traffic can be translated with minimum TCAM use. While translating these packets additional rules required to maintain consistency are added. Authors in Hua et al. (2016) proposed an ordered update approach using stamps and digital signatures for maintaining consistency even in adversarial settings. However, above solutions do not achieve a consistent solution for an update to be made. The paper (Brandt et al., 2016) studied the process of consistent migration of flows during a dynamic update. The authors proved that the deciding which flows to migrate is an NP hard problem and proposed an approximate solution which runs in polynomial time.

5.4. Traffic measurements

Various network management tasks such as traffic engineering, anomaly detection, depend on efficient network monitoring. In an operational network, a large number of packet flows occur at each instant. Monitoring of such flows at the data plane requires a large number of resources of the data plane devices including CPU and memory (Moshref et al., 2013, 2015). The monitored results are then periodically (or event based) sent to the controller. It utilizes already constraint network bandwidth. Therefore, monitoring all flows is a wastage of resources of data plane and the network bandwidth. Only the flows that affect the network are required for further operations at controller must be monitored (Yu et al., 2014; Gong et al., 2015; Liu et al., 2016a; Gibb et al., 2012; Sun et al., 2015a).

- *Measurements using traffic differentiation:* A memory-efficient system for flow monitoring can be achieved by differentiating which flows to monitor and which flows require direct forwarding without maintaining any measurements. Authors in Yu et al. (2014) proposed a distributed yet collaborative monitoring system. It uses Bloom filters to store monitoring rules within a limited memory. Separate monitoring tables, called admission Bloom filter (admBF), containing the flows that are to be monitored. These flows are stored on the controller and required switches along with the rule action to be taken if the flow is encountered. When a new packet reafichera2015ches the switch, it is matched against the monitoring rules. If it matches the admBF, it signifies that it is required to be monitored and sent to the next stage. Whereas if it does not match any admBF, the packet is directly sent to the flow table for processing. Another flow differentiation mechanism using temporal coordination information was proposed in the paper (Liu et al., 2016a). It selects few network devices as selectors, that selects which flows are needed to be measured and monitored, that perform the actual measurement. The selectors and monitors are required to be well coordinated for efficient flow monitoring. Characteristics achieved by the proposed method include coordinating the selector and the monitor devices timely by sending the monitoring message before the threshold is triggered. Using packet tagging while sending a message from the selector to the monitors, thus minimizing the bandwidth overhead and optimizing joint placement of selectors and monitors. It is based upon various constraints such as resource utilization, latency, etc.

The paper (Gibb et al., 2012) discussed advantages of outsourcing functionalities to different service providers. However, the details about what traffic is carried by which service provider is harder to identify and manage. The authors in Sun et al. (2015a) proposed a method that controls traffic from the edge router by dividing the total IP address space among various service providers. Each new traffic flow is mapped to a service provider by using source network address translation based on the destination of the traffic. This ensures that the traffic will utilize the specified service provider and traffic from each can now be separately monitored. Division of address space can be altered based on dynamic traffic conditions to optimize the traffic flow within the network.

- *Measurements using additional tool:* The traditional network uses the tool trace-route for identifying the paths in the network. The paper (Agarwal et al., 2014) presented a tool called SDN trace-route that allows operator to query the path taken by any packet through the SDN. It uses a probe packet with user-defined header fields launched in the network. At a certain point and report path as a list of ports on switches. To differentiate the trace packet from network traffic specific tags are used in the trace packet headers. SDN trace-route runs an application at the controller and collects the topology information. It is used while calculating the path from the obtained nodes. The Graph coloring problem is an NP-hard problem thus a greedy algorithm is used for minimizing the number of colors (tags) required for covering the network. Another software tool proposed for traffic measurement under shared dynamic network behavior is GitFlow (Dwaraki et al., 2015). It manages switch flow, provides data plane security, and flexibility in the control plane. A server co-located with the controller is used to keep the updated information about the flow in the data plane. A background module is used to

Table 15
Existing Challenges and solution of traffic measurement in SDN network.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Trevisan et al. (2018)	Per serviceâ management of traffic	Defining policies based on services	–	OpenFlow	Simulated	–	Traffic Management
Aujla et al. (2018b)	Secure Communication in IioT	Cuckoo filter based forwarding scheme	–	–	Testbed	–	Traffic Differentiation
Aslan and Matrawy (2016)	Monitor the network state in SDN	Mathematical model for load balancing	POX	OpenFlow	Simulated	–	Additional tool
Zhang et al. (2016)	End to end network tomography	Combining flow measurement and end-to-end performance tomography	–	OpenFlow	Testbed	CAIDA	Additional tool
Polverini et al. (2016)	Traffic matrix estimation	Flow spread-based algorithm	–	–	Simulated	Nobel	Additional tool
Xu et al. (2017b)	Flow statistics collection in data plane	Approximation algorithm based on randomized rounding	Ryu	OpenFlow	Testbed	–	Traffic measurements
Yu et al. (2014)	Overload of monitoring tasks	Hash functions, wildcard rule lookup, and Bloom filters	–	OpenFlow	Simulated	EDU1, Fat-trees	Traffic Differentiation
Liu et al. (2016a)	Temporal coordination of measurement	Selective monitoring of flows	–	OpenFlow	Simulated	B4, CAIDA	Traffic Differentiation
Sun et al. (2015a)	Inbound traffic engineering	Source network address translation	–	OpenFlow	Testbed	Star	Traffic Differentiation
Agarwal et al. (2014)	Network troubleshooting	SDN trace route	Floodlight	OpenFlow	Simulated	–	Additional tool
Dwaraki et al. (2015)	Flow state revisioning	Run a repository server to maintain flow state	–	–	Theoretical	–	Additional tool
Taylor et al. (2016)	Obtain flow-level information of the traffic	Shift agents from the network to end hosts	–	OpenFlow	Simulated	–	Additional tool
Gibb et al. (2012)	Outsourcing in-network features	Resource Outsourcing	Beacon	OpenFlow	Testbed	–	Traffic Differentiation

monitor the flow modification messages from the controller to the switches. Every changes are updated on the Git server and can be read by any application when required. A conflict resolution algorithm is also proposed. The model however has been tested only for single table switches.

In all the above mentioned studies the traffic at the switches is measured without any details of the network hosts operating on the network. The paper (Taylor et al., 2016) provided network operators with detailed visibility of the end hosts. So, as to scalable flow level information for all network traffic. Host agents are established for collecting the context of host and monitoring the traffic from it. The collected information is sent to the controller on demand. Host agents can optimally filter traffic at the source host to avoid network overhead. A summary of all the work related to traffic measurement in SDN is shown in Table 15.

6. Network verification

The network verification is a process to identify whether the designed network is suitable for providing the desired QoS. It commonly improves the network performance and security. Network operators need sophisticated tools and techniques to proactively catch these bugs before they can occur in production network. Previous work in this area had focused on verifying static network invariants over dynamically changing network (Beckett et al., 2014). Different from the traditional network, the verification of SDN consists various challenges because of dynamic changes in network states. This section focuses on verification methods of SDN and their challenges, a summary of which is presented in Table 16.

6.1. Hardware testing

Centralized control over the network in SDN allows policies to be applied and also tested at the global network level. The policies and applications defined in the network changes. The configuration of a given network element and the topology of the network may change because of device failure. Thus, SDN testing must address how to determine the correctness of a network and its components in dynamic configuration. Challenges addressed in network testing are summarized in Table 17. Network testing can be performed at different levels as defined and discussed ahead.

6.1.1. Performing testing at data plane

Network can be tested for faulty links and nodes by installing required rules in the switches. The approach proposed in paper (Kozat et al., 2014) aims to install an optimal or near-optimal number of static forwarding rules on switches/routers. For any controller to be able to verify the topology connectivity and detect link failures at data plane. It does not rely on state updates from forwarding plane nodes and other controllers while requiring reachability to only one (arbitrary) forward-

Table 16
Network verification of SDN.

Network Verification	Testing	Performing testing at Data Plane (Kozat et al., 2014; Perešíni et al., 2015; Yao et al., 2014; Fayaz et al., 2016) Performing Testing at Global Level (Kuzniar et al., 2014; Gupta et al., 2013; Fayazbakhsh et al., 2013; Ghasemi et al., 2017; Xu et al., 2016) Performing Testing at Intermediate Level (Khurshid et al., 2013; Xu et al., 2015)
	Debugging	Heller et al. (2013), Gupta et al. (2013), Lévai et al. (2015), Handigol et al. (2012), Durairajan et al. (2014) and Wundsam et al. (2011)
	Security	Malicious Controllers and Applications (Othman and Okamura, 2013; Matsumoto et al., 2014; Wen et al., 2013; Scott-Hayward et al., 2014; Porras et al., 2012; Chandrasekaran and Benson, 2014; Afek et al., 2017) Denial of Service (Braga et al., 2010; Liu et al., 2017; Fichera et al., 2015) Firewall in Dynamic Network (Wang et al., 2013; Hu et al., 2014; Vörös and Kiss, 2016)

Table 17
Testing Mechanism of SDN its existing challenges and solutions.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Sun et al. (2015b)	Application verification under OpenFlow conditions	Mitigate ill effects of race conditions	NOX	OpenFlow	Simulated	–	Intermediate level
Kozat et al. (2014)	Verifying connectivity	Graph based approach	–	–	Simulated	–	Data plane level
Perešíni et al. (2015)	Identifying switch/link failure	Packet probing	–	–	Simulated	Internet Topology Zoo	Data plane level
Bu et al. (2016)	Identifying forwarding faults	Packet probing	Ryu	OpenFlow	Prototype	–	Data plane level
Yao et al. (2014)	Dynamic forwarding in multiple flow tables	Extended state machine model	–	OpenFlow	Simulated	–	Data plane level
Fayaz and Sekar (2014)	Testing dynamic policies on switches	State machine model	POX	OpenFlow	Simulated	–	Data plane level
Fayaz et al. (2016)	Data plane modeling	State machine model	Opendaylight	–	Simulated	Internet Topology Zoo	Data plane level
Gupta et al. (2013)	Simulator design	fs flow level simulator	POX	OpenFlow	Simulated	Linear	Global level
Fayazbakhsh et al. (2013)	Flow monitoring in presence of middle-boxes	Flow tagging	–	OpenFlow	Simulated	–	Global level
Khurshid et al. (2013)	Update verification	Middleware between data and control plane	–	OpenFlow	Simulated	Rocketfuel	Intermediate level
Xu et al. (2015)	Consistency verification	Graph based state extraction	–	–	Prototype	–	Intermediate level

ing node. Though there is an overhead of traversing some links more than once. Hence, this may increase latency, but it helps in reducing the number of static forwarding rules. Another method of data plane testing is by using probe packets. The authors in Perešíni et al. (2015) proposed a system which identifies data plane faults by systematically probing the switch data plane. The forwarding logic is modeled as a boolean satisfiability problem for generating probes. The Probes allow monitoring of both static and dynamic flow tables. Another work that relies on probing functionalities for identifying the faults in switches is RuleScope. The paper RuleScope (Wen et al., 2017) presented various algorithms for identifying forwarding faults on data plane. It will successfully finds missing faults (when a rule is not active on a switch) and priority faults (when overlapping rules violate the priority order).

Stateful and dynamic data functions cannot be tested by matching the header of a packet within the flow. They require to test a sequence of packets in a flow or a chain of network state transitions in the data plane. Authors in the paper (Yao et al., 2014) proposed a model based black box testing scheme. A Pipelined Extended Finite State Machine (Pi-EFM) is presented that can model SDN data plane with multiple flow tables. Pi-EFM is formed by modifying extended finite state machines. The proposed model reduces the risk of state explosion and provides systematic coverage to all the components. Similar work utilizing the state machine model proposed in Fayaz and Sekar (2014). It uses a conceptual testing tool, FlowTest for testing data plane handling dynamic network policies. First data plane functions to be tested are modeled as state machines. These state machines are then combined to form an overall test model of the network. Given the initial states and the network model test plans can be generated by creating a sequence of transitions to move the network to a required state. AI planning tools were used for generating transitions sequences. State machine model can also be used for testing of network implementing complex context-dependent policies. Authors in Fayaz et al. (2016) used high-level abstraction for modeling data plane as a sequence of finite-state machines.

6.1.2. Performing testing at global level

Application and state verifications in SDN need to be performed at global network level. Challenges faced while testing the efficiency of SDN applications at global level include complex large size network and difficulty in realizing the test environments for the network (Kuzniar et al., 2014; Gupta et al., 2013; Fayazbakhsh et al., 2013; Ghasemi et al., 2017; Xu et al., 2016).

A tool Fs-SDN is proposed in Gupta et al. (2013) that can prototype the network and efficiently test SDN applications. The authors addressed the problem of prototyping and evaluating new SDN-based applications accurately, at large scale. In this way that enables easy translation to real controller platforms like POX and NOX. Another challenge encountered in testing at global level is due to the presence of the middle-boxes (*e.g.,* NATs, firewalls, intrusion detection systems, proxies). It is because they modify the packets, making it difficult for the controller to verify the realization of policies in the network. The authors in Fayazbakhsh et al. (2013) proposed a solution which is using FlowTags associated with every policy, such that even when headers and packets are modified the tags can be used for identifying the flows. The packets while being processed by middle-boxes are tagged to include the context of the modifying middle-box. These tags are then considered by the switches and other middle-boxes as part of header and thus affect the policy enforcement. Tag based solutions are however not scalable.

6.1.3. Performing testing at intermediate level

Testing how the application policies affect the underlying network can also be performed at intermediate layer lying between the control plane and the data plane. Authors in Khurshid et al. (2013) proposed a design, called VeriFlow, that allows real-time verification of network

updates. The VeriFlow, implemented as a layer between the control plane and data plane, intercepts all messages between the controller and the devices. When it identifies any rule modifying event it initiates the rule verification module. The network is partitioned into equivalence classes based on the fact that they are affected by same forwarding rules. Thus, for any modification only a subset of the network needs to be checked.

Growing cloud environments such as OpenStack demands edge based SDN that can configure end hosts. This will ensure consistency between the high level network policies and low level devices state is a challenging task. To identifying any inconsistency that occur, especially, in edge SDN is also challenging. Inconsistencies in SDN arise due to unreliable and asynchronous communication between controller and hosts and hardware and software errors (Xu et al., 2015). State information is periodically extracted from the controller and hosts. It is parsed and verified by the verification server placed on the controller. Graph based approach is used for state extraction that stores header fields along with required action in boolean bit format. The State verification technique of L2-L4 layer is defined in the graph based approach. The proposed method however, can only identify inconsistencies in network by comparing snapshots of network. It did not work in real-time network.

6.2. Debugging

Faults within a network can occur at any point and can be caused by operators or external users or by improper functioning of any hardware or software devices. With such wide variety of causes, a complex and large network is thus difficult to debug. The network administrators thus require an automatic debugger that can detect the error, its location of fault and provide measures to fix it.

The presence of programmatic control over the network allows the programmers to use software like debuggers to detect network faults such as overlapping rules, rule conflicts, flow loops etc. Authors in Heller et al. (2013) shown that a centralized SDN controller with its global network view can be leveraged to design an automatic debugger. The SDN is divided into following state layers based on configuration: policy, logical view, physical view, device state, and hardware. A fault in a network, usually occurs when there is a mismatch between the states of two consecutive layers. The paper, however does not mention any measure for fault recovery and neither provide any implementation results. The paper (Lévai et al., 2015) also leveraged the layered approach for identifying software bugs can occur all over the SDN stack.

The authors in Handigol et al. (2012) proposed another debugger, known as network debugger (NBD), which implements breakpoints and packet back-traces for debugging the SDN. A packet back-trace in NBD defines a packet breakpoint (*e.g.*, an un-forwarded packet or a packet filter), then shows the sequence of forwarding actions faced by that packet leading to the breakpoint. The information in a packet back-trace can be used by SDN programmers to resolve logical errors in the control program. It help in identifying any hardware related faults. The paper (Durairajan et al., 2014) presented the SDN debugger built over fs-SDN (Gupta et al., 2013). A similar diagnostic tool is OFRewind (Wundsam et al., 2011) that also records network events. These records can be replayed later for troubleshooting.

The stateful and dynamic nature of application policies (*e.g.*, dynamic access control) was considered in Beckett et al. (2014). The authors proposed an assertion language that can verify the dynamic network behavior to find bugs in the network. This language allows programmers to design their applications while using their assertion statements. It can check errors based on the policy that is required to be designed. The dynamic variations in the network can cause occurrence of bugs. This is explained by a MAC learning switch and stateful firewall example. It also discusses that if the assertions are used at the time of designing the applications. Bugs in the above examples can be avoided. Assertion primitives provided to the programmers are assert-now that checks the assertion at the exact event where it is defined. Assert-continuously and stop commands to check the property that is required to be maintained during the defined interval. The paper (Miserez et al., 2015) presented techniques for automatically detecting concurrency errors in dynamic SDN environment. It identifies discrepancies between a control program and the underlying network using a happens-before relation (Lamport, 1978). Table 18 shows the summary of existing work related to debugging in SDN.

6.3. Security

The SDN shifts traditional networking control from hardware to software. This change provides simplification in network operations and administration. Work of network designers shifts from coding low-level device configurations to designing software that can provide network management and debugging facility. On the down-side, the decoupled design of SDN has raised additional security challenges in the network. The programing ability and centralized control features of SDN introduce new areas of faults and attacks. The authors in Kreutz et al. (2013) are discussed various threat vectors available in SDN. The authors proposed a platform design for minimizing the threats or at-least provide fault-tolerance in the network. Measures incorporated in the network design are replication of applications and controller instances, diversity in control platform, dynamic device association achieved by associating switches with more than one controller, ensuring trust between devices and controllers and between controllers and applications. Fast and reliable software update, to reduce the vulnerable time. Fig. 18 illustrates different security issues that arise in the SDN and summary of their existing work is presented in Table 19.

6.3.1. Malicious controllers and applications

Multiple controllers in a distributed SDN concurrently access the data plane of the network. Similarly, applications from a different network provides may access a pool of controllers. An attacker can easily gain access to network resources and manipulate the network operation if the attacker impersonate the controllers or applications (Othman and Okamura, 2013; Matsumoto et al., 2014; Wen et al., 2013; Scott-Hayward et al., 2014; Porras et al., 2012; Chandrasekaran and Benson, 2014). The attacker can misconfigured the controllers and degrades the network performance.

The authors in Othman and Okamura (2013) proposed a system to ensure security in distributed controllers where the central control elements are secured by transport layer security. The system uses a centralized trust manager and a signature checking scheme. It thus introduces high overhead in generation of communication messages. Threat of unauthorized access can also be minimized by a hierarchical system of controllers in Wen et al. (2013). The root controller in the proposed work is responsible for all global decisions. This approach assumes that the root controller is trusted which cannot always be guaranteed. The authors in Matsumoto et al. (2014) proposed a controller architecture, called Fleet. It tries to solve the problem of unauthorized access among the network administrators by using digital signatures. The Fleet defines a switch intelligence layer that contains an instance of each switch on which the configuration decided by the administrators is placed. This layer verifies the configuration and installs the related rules on the switches. The authors in Scott-Hayward et al. (2014) shown that the network access to different applications can also be restricted by using a set of permissions and an isolation mechanism. This work suggested that API entry is the suitable location for enforcing the permissions.

Entry of malicious applications in the network can be avoided if the controller authenticates every application. FortNOX presented a rule-conflict detection module in Porras et al. (2012) that verifies all the rule insertion requests before they are applied in the network. It implements role-based authentication for determining the security authorization of applications defining the rules. The rule-conflicts are analyzed by an

Table 18
Existing challenges and solutions for debugging in SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Perešíni et al. (2018)	Diagnosing Network Anomalies	Boolean satisfiability formulation	–	OpenFlow	Testbed	–	Data plane level
Beckett et al. (2014)	Dynamic network verification	Assertion language	POX	OpenFlow	Simulation	Linear	All planes
Heller et al. (2013)	Debugger	Layered approach	Ryu	OpenFlow	Testbed	–	All planes
Lévai et al. (2015)	Debugger	Layered approach	–	–	Theoretical	–	All planes
Handigol et al. (2012)	Debugger	Breakpoint and packet backtrace	–	–	Prototype	–	All planes
Durairajan et al. (2014)	Packet flow debugger	Stepping, Break points and watch variable	–	–	Prototype	–	All planes
Wundsam et al. (2011)	Dynamic debugging	Record network	POX	OpenFlow	Simulation	–	All planes
Miserez et al. (2015)	Detecting concurrency errors	Happen-before model	Floodlight, POX	OpenFlow	Prototype	–	All planes

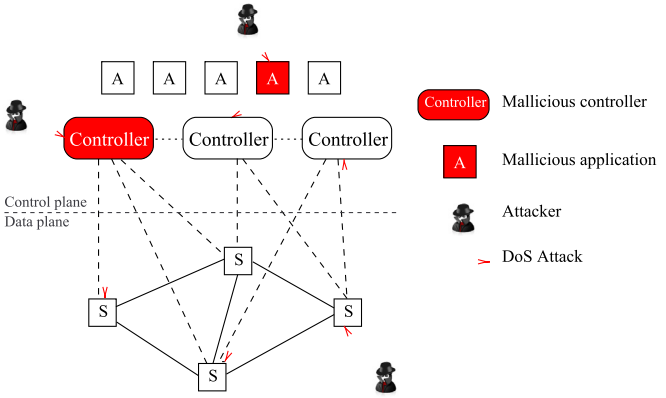


Fig. 18. Security threats on SDN.

algorithm called alias set rule reduction that detects rule contradictions in the presence of dynamic flow.

Authors in Afek et al. (2017) suggested that the current SDN match and action model is good at handling the anti-spoofing method. Further, the authors used advance dynamic resource sharing for distribution of resources over the network. For successful operation, count of flow table rules and switch to controller messages, must be proportional to the traffic of the network.

6.3.2. Denial of service

Denial of Services (DoS) security attack heavily degrades the performance of SDN (Sezer et al., 2013; Chung et al., 2013; Jeong et al., 2014). A large number of packets in DoS attack are sent in the network with varying IP addresses. As the rules related to these packets do not exist in the switches they are forwarded to the controller leading to exhaustion of its resources. Thus making it unavailable for valid switch requests (Braga et al., 2010; Liu et al., 2017). A similar DoS attack can be launched at the data plane by flooding the flow table of the switches (Kuerban et al., 2016). Threat of DoS attack can be minimized by utilizing moving target defense methodology. The authors in Jafarian et al. (2012) and Kampanakis et al. (2014) used a moving target technique, called the OpenFlow random host mutation. It frequently and randomly changes IP addresses of the host. The technique aims at performing mutation without the knowledge of the hosts and at such speed and randomness that cannot be tracked by the attacker. The dynamic IP address binding is used to solve the problem of IP spoofing. The proposed C-SVM algorithm is used for identifying attacks and required rules for blocking the attack are added to flow tables at the source switch. SDN can handle the failure from a single control point, which is having complete network information. The attack could be launched directly on this single control point, *i.e.,* controller which is a unique failure point. To overcome this problem in failure handling authors in Fichera et al. (2015) proposed OpenFlow based scheme for mitigating TCP SYN FLOOD attack. This will ensure smooth deployment of security features and mitigate single point failure.

6.3.3. Firewall in dynamic network

Building a robust firewall is another security challenge in SDN. Dynamic network states make the development of a firewall impractical in SDN. It is also true that the stateless data plane that does not stores flow states cannot perform flow verification. Therefore, the implementation of firewall in SDN consists various challenges. The authors in Wang et al. (2013) used header space analysis to develop a conflict detection and resolution algorithm for SDN firewall. Header space analysis generated a graph model that shows the possible paths traversed by the packet. The main limitation of this work is that it considers only the flow re-write actions for bypassing the threats. Another work using the header space analysis is FlowGuard (Hu et al., 2014), that act

Table 19
Existing challenges and solutions for security in SDN.

Paper	Address the challenge	Proposed solution	Controller	Southbound API	Validated	Topology	Technique Used
Aujla et al. (2018b)	Secure Communication in IioT	Kerberos based authentication	–	OpenFlow	Testbed	–	–
Luo et al. (2016)	Multi stage attack in Software-defined Home Networks	Multi-stage attack mitigation mechanism	–	–	Simulated	–	Firewall
Cui et al. (2016)	Security in SDN	Fingerprinting the controller-switch interactions	Floodlight	OpenFlow	Testbed	–	Malicious applications
Lara and Ramamurthy (2016)	Security in SDN	Policy based security	Floodlight	OpenFlow	Simulated	GENI	Firewall
Sun et al. (2015c)	Net heterogeneity in mobile networks	Integrate SDN, NFV and SDR	–	–	Theoretical	–	Malicious applications
Deng et al. (2018)	Packet Injection Attack	PacketChecker, extension module	Floodlight	OpenFlow	Simulated	Fat-Trees	Spoofing, DoS
Othman and Okamura (2013)	Security in distributed controllers	TLS in control plane	–	OpenFlow	Theoretical	–	Malicious applications
Wen et al. (2013)	Unauthorized access	Hierarchical system of controllers	–	OpenFlow	Theoretical	–	Malicious applications
Matsumoto et al. (2014)	Unauthorized access	Digital Signatures	POX	OpenFlow	Simulated	Random	Malicious applications
Scott-Hayward et al. (2014)	Network access by different apps	Isolation mechanism	Floodlight	OpenFlow	Simulated	–	Malicious applications
Porras et al. (2012)	Rule conflicts	Role based authentication	NOX	OpenFlow	Testbed	–	Malicious applications
Jafarian et al. (2012)	DoS attack	Moving Target technique	NOX	OpenFlow	Simulated	–	DoS
Kampanakis et al. (2014)	DoS attack	Moving Target technique		OpenFlow	Simulated	–	DoS
Liu et al. (2017)	IP Spoofing	Dynamic IP address binding using SVM	Floodlight	OpenFlow	Simulated	–	DoS
Wang et al. (2013)	Firewall in SDN	Header space analysis	–	–	–	–	Firewall
Hu et al. (2014)	Firewall in SDN	Header space analysis	Floodlight	OpenFlow	Testbed	Stanford backbone	Firewall
Fichera et al. (2015)	TCP SYN FLOOD attack	OpenFlow based scheme for failure mitigation	–	OpenFlow	Simulated	–	DDoS
Afek et al. (2017)	Network anti-spoofing	Advance dynamic resource sharing	POX	OpenFlow	Simulated	–	Malicious applications
Vörös and Kiss (2016)	Security programming	Protocol-independent Packet Processors	–	OpenFlow	Simulated	–	Firewall
Varadharajan et al. (2019)	Fine grained security policies	Policy driven security architecture	–	OpenFlow	Simulated	Zoo	Malicious application

as a robust firewall even in dynamic network conditions. It first uses header space analysis for flow tracking. Next, a Firewall Authorization Space (FAS) is calculated, based on the firewall policy requirement, that represent those address ranges that are authorized by the policy operators. Violation of firewall rules is detected by matching the flow space against the FAS. Advantage in FlowGuard is that it allows flow rejecting, dependency breaking, update rejecting, flow removing and packet blocking when a mismatch occurs. Authors in Vörös and Kiss (2016) addressed the problem associated with Openflow, of not supporting new header definition. Since new header definition is necessary for network packet encapsulation, so authors proposed a high-level language defined as protocol independent packet processing, for adding an extra header in the network packet. This high-level language will resolve above shortcoming of the OpenFlow and provide ease of adding security features.

7. Conclusion

In this paper, we have presented a comprehensive survey of various research areas in SDN design, implementation and performance. Based upon the SDN challenges, issues, and research directions, the study focuses on network design of SDN, its implementation, performance evaluation and finally its verification. The network design emphasized on the scalability of SDN, its fault-tolerance, flexibility, and elasticity. On the other hand, network implementation focuses on integration with traditional network, resource management in SDN and its virtualization. The limited resources in SDN are the bandwidth between switches and controller and the memory which must be efficiently utilized. After successful designing of SDN and its implementation, network performance is evaluated based on latency, efficiency and consistency. Finally, the network verification part is covered, illustrating the testing, debugging and security issues in the SDN.

The comprehensive survey of SDN clearly identifies the several research possibilities in SDN, although several works had been done in the area of SDN but still a lot more things to be uncovered. The work done in this area is still in its initial stage. The future work in the SDN leads to increase acceptance and decrement in the cost of setting the network. Along with cost reduction, security in SDN is one of the more challenging issues in the SDN which must emphasize in future research.

Acknowledgements

This work is supported by SERB file ECR/2016/000406/ES and scheme Early Career Research Award.

References

Abolhasan, M., Lipman, J., Ni, W., Hagelstein, B., 2015. Software-defined wireless networking: centralized, distributed, or hybrid? IEEE Netw. 29 (4), 32–38.
Afek, Y., Bremler-Barr, A., Shafir, L., 2017. Network anti-spoofing with sdn data plane. In: Proc. of IEEE INFOCOM. IEEE, pp. 1–9.
Agarwal, K., Rozner, E., Dixon, C., Carter, J., 2014. SDN traceroute: tracing SDN forwarding without changing network behavior. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 145–150.
Aguado, A., Davis, M., Peng, S., Alvarez, M.V., Lpez, V., Szyrkowiec, T., Autenrieth, A., Vilalta, R., Mayoral, A., Muoz, R., et al., 2016. Dynamic virtual network reconfiguration over sdn orchestrated multitechnology optical transport domains. J. Light. Technol. 34 (8), 1933–1938.
Al-Shabibi, A., De Leenheer, M., Gerola, M., Koshibe, A., Parulkar, G., Salvadori, E., Snow, B., 2014. OpenVirteX: make your virtual SDNs programmable. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 25–30.
Amokrane, A., Langar, R., Boutaba, R., Pujolle, G., 2015. Flow-based management for energy efficient campus networks. IEEE Trans. Netw. Serv. Manag. 12 (4), 565–579.
Anadiotis, A.C., Morabito, G., Palazzo, S., 2016. An SDN-assisted framework for optimal deployment of MapReduce functions in WSNs. IEEE Trans. Mob. Comput. 15 (9), 2165–2178.
Aslan, M., Matrawy, A., 2016. On the impact of network state collection on the performance of sdn applications. IEEE Commun. Lett. 20 (1), 5–8.
Astaneh, S.A., Heydari, S.S., 2016. Optimization of sdn flow operations in multi-failure restoration scenarios. IEEE Trans. Netw. Serv. Manag. 13 (3), 421–432.
Aujla, G.S., Kumar, N., Zomaya, A.Y., Ranjan, R., 2018a. Optimal decision making for big data processing at edge-cloud environment: an sdn perspective. IEEE Trans. Ind. Inf. 14 (2), 778–789.
Aujla, G.S., Chaudhary, R., Garg, S., Kumar, N., Rodrigues, J.J., 2018b. Sdn-enabled multi-attribute-based secure communication for smart grid in iiot environment. IEEE Trans. Ind. Inf. 14 (6), 2629–2640.
Azodolmolky, S., Nejabati, R., Escalona, E., Jayakumar, R., Efstathiou, N., Simeonidou, D., 2011. Integrated openflowgmpls control plane: an overlay model for software defined packet over optical networks. Optic Express 19 (26), B421–B428.
Baldin, I., Huang, S., Gopidi, R., 2014. A resource delegation framework for SDNs. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 49–54.
Bao, K., Matyjas, J.D., Hu, F., Kumar, S., 2018. Intelligent software-defined mesh networks with link-failure adaptive traffic balancing. IEEE Trans. Cognit. Commun. Netw. 103 (1), 14–76.
Basta, A., Blenk, A., Hoffmann, K., Morper, H.J., Hoffmann, M., Kellerer, W., 2017. Towards a cost optimal design for a 5g mobile core network based on sdn and nfv. IEEE Trans. Netw. Serv. Manag. 14 (4), 1061–1075.
Beckett, R., Zou, X.K., Zhang, S., Malik, S., Rexford, J., Walker, D., 2014. An assertion language for debugging SDN applications. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 91–96.
Belbekkouche, A., Hasan, M.M., Karmouch, A., 2012. Resource discovery and allocation in network virtualization. IEEE Commun. Surv. Tutor. 14 (4), 1114–1128.
Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., et al., 2014. Onos: towards an open, distributed sdn os. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 1–6.
Bianchi, G., Bonola, M., Capone, A., Cascone, C., 2014. Openstate: programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Comput. Commun. Rev. 44 (2), 44–51.
Bidkar, S., Gumaste, A., Ghodasara, P., Hote, S., Kushwaha, A., Patil, G., Sonnis, S., Ambasta, R., Nayak, B., Agrawal, P., 2014. Field trial of a software defined network (sdn) using carrier ethernet and segment routing in a tier-1 provider. In: Proc. of IEEE GLOBECOM, pp. 2166–2172.
Bifulco, R., Boite, J., Bouet, M., Schneider, F., 2016. Improving sdn with inspired switches. In: Proc. of ACM Symposium on SDN Research, pp. 1–12.
Blenk, A., Basta, A., Zerwas, J., Reisslein, M., Kellerer, W., 2016. Control plane latency with sdn network hypervisors: the cost of virtualization. IEEE Trans. Netw. Serv. Manag. 13 (3), 366–380.
Bondi, A.B., 2000. Characteristics of scalability and their impact on performance. In: Proc. of ACM International Workshop on Software and Performance, pp. 195–203.
Borokhovich, M., Schiff, L., Schmid, S., 2014. Provable data plane connectivity with local fast failover: introducing openflow graph algorithms. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 121–126.
Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., Horowitz, M., 2013. Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. In: Proc. of ACM SIGCOMM Computer Communication Review, vol. 43, pp. 99–110.
Botelho, F., Ramos, F.M.V., Kreutz, D., Bessani, A., 2013. On the feasibility of a consistent and fault-tolerant data store for sdns. In: Proc. of IEEE EWSDN, pp. 38–43.
Bouten, N., Mijumbi, R., Serrat, J., Famaey, J., Latr, S., Turck, F.D., 2017. Semantically enhanced mapping algorithm for affinity-constrained service function chain requests. IEEE Trans. Netw. Serv. Manag. 14 (2), 317–331.
Braga, R., Mota, E., Passito, A., 2010. Lightweight ddos flooding attack detection using nox/openflow. In: Proc. of IEEE LCN, pp. 408–415.
Brandt, S., Frster, K.-T., Wattenhofer, R., 2016. On consistent migration of flows in SDNs. In: Proc. of IEEE INFOCOM, pp. 1–9.
Bremler-Barr, A., Harchol, Y., Hay, D., 2016. Openbox: a software-defined framework for developing, deploying, and managing network functions. In: Proc. of ACM SIGCOMM Conference, pp. 511–524.
Bruschi, R., Davoli, F., Lago, P., Lombardo, A., Lombardo, C., Rametta, C., Schembra, G., 2017. An sdn/nfv platform for personal cloud services. IEEE Trans. Netw. Serv. Manag. 14 (4), 1143–1156.
Bu, K., Wen, X., Yang, B., Chen, Y., Li, L.E., Chen, X., 2016. Is every flow on the right track?: inspect sdn forwarding with rulescope. In: Proc. of IEEE INFOCOM, pp. 1–9.
Cai, Y., Yu, F.R., Liang, C., Sun, B., Yan, Q., 2016. Software-defined device-to-device (d2d) communications in virtual wireless networks with imperfect network state information (nsi). IEEE Trans. Veh. Technol. 65 (9), 7349–7360.
Caria, M., Jukan, A., Hoffmann, M., 2016. Sdn partitioning: a centralized control plane for distributed routing protocols. IEEE Trans. Netw. Serv. Manag. 13 (3), 381–393.
Casado, M., Koponen, T., Shenker, S., Tootoonchian, A., 2012. Fabric: a retrospective on evolving SDN. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 85–90.
Cascone, C., Pollini, L., Sanvito, D., Capone, A., 2015. Traffic management applications for stateful sdn data plane. In: Proc. of IEEE EWSDN, pp. 85–90.
Casellas, R., Martnez, R., Muoz, R., Vilalta, R., Liu, L., Tsuritani, T., Morita, I., 2013. Control and management of flexi-grid optical networks with an integrated stateful path computation element and openflow controller. J. Opt. Commun. Netw. 5 (10), A57–A65.
Casellas, R., Muoz, R., Martnez, R., Vilalta, R., Liu, L., Tsuritani, T., Morita, I., Lpez, V., de Dios, O.G., Fernndez-Palacios, J.P., 2015. Sdn orchestration of openflow and gmpls flexi-grid networks with a stateful hierarchical pce. J. Opt. Commun. Netw. 7 (1), A106–A117.
Cello, M., Marchese, M., Mongelli, M., 2016. On the qos estimation in an openflow network: the packet loss case. IEEE Commun. Lett. 20 (3), 554–557.
Chandrasekaran, B., Benson, T., 2014. Tolerating sdn application failures with legosdn. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 1–7.

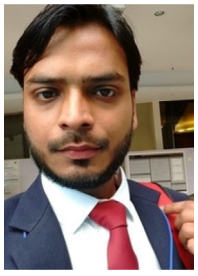
Channegowda, M., Nejabati, R., Fard, M.R., Peng, S., Amaya, N., Zervas, G., Simeonidou, D., Vilalta, R., Casellas, R., Martnez, R., et al., 2013. Experimental demonstration of an openflow based software-defined optical network employing packet, fixed and flexible dwdm grid technologies on an international multi-domain testbed. Optic Express 21 (5), 5487–5498.
Chau, N.-T., Nguyen, M.-D., Jung, S., Jung, S., 2014. Secaas framework and architecture: a design of dynamic packet control. In: Proc. of Springer International Workshop on Information Security Applications, pp. 190–201.
Chen, C., Liu, C., Liu, P., Loo, B.T., Ding, L., 2015. A scalable multi-datacenter layer-2 network architecture. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–12.
Chen, J., Chen, J., Ling, J., Zhang, W., 2016a. Failure recovery using vlan-tag in sdn: high speed with low memory requirement. In: Proc. of IEEE IPCCC, pp. 1–9.
Chen, J.L., Ma, Y.W., Kuo, H.Y., Yang, C.S., Hung, W.C., 2016b. Software-defined network virtualization platform for enterprise network resource management. IEEE Trans. Emerg. Top. Comput. 4 (2), 179–186.
Chen, C., Li, D., Li, J., Zhu, K., 2016c. Svdc: a highly scalable isolation architecture for virtualized layer-2 data center networks. IEEE Trans. Cloud Comput. PP (99), 1.
Chen, F., Wu, C., Hong, X., Lu, Z., Wang, Z., Lin, C., 2016d. Engineering traffic uncertainty in the openflow data plane. In: Proc. of IEEE INFOCOM, pp. 1–9.
Chowdhury, N.M.K., Boutaba, R., 2009. Network virtualization: state of the art and research challenges. IEEE Commun. Mag. 47 (7), 20–26.
Chung, C.-J., Khatkar, P., Xing, T., Lee, J., Huang, D., 2013. Nice: network intrusion detection and countermeasure selection in virtual network systems. IEEE Trans. Dependable Secure Comput. 10 (4), 198–211.
Cohen, R., Lewin-Eytan, L., Naor, J.S., Raz, D., 2014. On the effect of forwarding table size on sdn network utilization. In: Proc. of IEEE INFOCOM, pp. 1734–1742.
Coronado, E., Riggio, R., Villaln, J., Garrido, A., 2018. Joint mobility management and multicast rate adaptation in softwaredefined enterprise wlans. IEEE Trans. Netw. Serv. Manag. 15 (2), 625–637.
Cugini, F., Paolucci, F., Fresi, F., Meloni, G., Sambo, N., Pot, L., D'Errico, A., Castoldi, P., 2016. Toward plug-and-play software-defined elastic optical networks. J. Light. Technol. 34 (6), 1494–1500.
Cui, H., Karame, G.O., Klaedtke, F., Bifulco, R., 2016. On the fingerprinting of software-defined networks. IEEE Trans. Inf. Forensics Secur. 11 (10), 2160–2173.
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, pp. 254–265.
Cziva, R., Jout, S., Stapleton, D., Tso, F.P., Pezaros, D.P., 2016. Sdn-based virtual machine management for cloud data centers. IEEE Trans. Netw. Serv. Manag. 13 (2), 212–225.
Dai, W., Shou, G., Hu, Y., Guo, Z., R-sdn, 2014. A recusive approach for scaling sdn. In: Proc. of IEEE ICT, pp. 1–6.
Das, S., Parulkar, G., McKeown, N., Singh, P., Getachew, D., Ong, L., 2010. Packet and circuit network convergence with openflow. In: Proc. of Optical Fiber Communication, pp. 1–3.
Das, D., Bapat, J., Das, D., 2017. A dynamic qos negotiation mechanism between wired and wireless sdn domains. IEEE Trans. Netw. Serv. Manag. 14 (4), 1076–1085.
Deng, S., Gao, X., Lu, Z., Gao, X., 2018. Packet injection attack and its defense in software-defined networks. IEEE Trans. Inf. Forensics Secur. 13 (3), 695–705.
Ding, W., Qi, W., Wang, J., Chen, B., 2015. Openscaas: an open service chain as a service platform toward the integration of sdn and nfv. IEEE Netw. 29 (3), 30–35.
Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., Kompella, R., 2013. Towards an elastic distributed SDN controller. In: ACM SIGCOMM Computer Communication Review, vol. 43, pp. 7–12.
Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., Kompella, R.R., 2014a. Elasticon; an elastic distributed sdn controller. In: Proc. of ACM/IEEE Symposium on ANCS, pp. 17–27.
Dixit, A., Kogan, K., Eugster, P., 2014b. Composing heterogeneous sdn controllers with flowbricks. In: Proc. of IEEE ICNP, pp. 287–292.
Dong, M., Li, H., Ota, K., Xiao, J., 2015. Rule caching in sdn-enabled mobile access networks. IEEE Netw. 29 (4), 40–45.
Dorsch, N., Kurtz, F., Girke, F., Wietfeld, C., 2016. Enhanced fast failover for software-defined smart grid communication networks. In: Proc. of IEEE GLOBECOM. IEEE, pp. 1–6.
Durairajan, R., Sommers, J., Barford, P., 2014. Controller-agnostic sdn debugging. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, pp. 227–234.
Dwaraki, A., Seetharaman, S., Natarajan, S., Wolf, T., 2015. Gitflow: flow revision management for software-defined networks. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–6.
Fang, L., Chiussi, F., Bansal, D., Gill, V., Lin, T., Cox, J., Ratterree, G., 2015. Hierarchical sdn for the hyper-scale, hyper-elastic data center and cloud. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–13.
Faraci, G., Schembra, G., 2015. An analytical model to design and manage a green sdn/nfv cpe node. IEEE Trans. Netw. Serv. Manag. 12 (3), 435–450.
Fayaz, S.K., Sekar, V., 2014. Testing stateful and dynamic data planes with FlowTest. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 79–84.
Fayaz, S.K., Yu, T., Tobioka, Y., Chaki, S., Sekar, V., 2016. Buzz: testing context-dependent policies in stateful networks. In: Proc. of the USENIX Symposium on NSDI, pp. 275–289.
Fayazbakhsh, S.K., Sekar, V., Yu, M., Mogul, J.C., 2013. FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 19–24.
Ferguson, A.D., Guha, A., Liang, C., Fonseca, R., Krishnamurthi, S., 2012. Hierarchical policies for SDNs. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 37–42.
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, pp. 327–338.
Fernndez-Fernndez, A., Cervell-Pastor, C., Ochoa-Aday, L., 2017. A multi-objective routing strategy for qos and energy awareness in software-defined networks. IEEE Commun. Lett. 21 (11), 2416–2419.
Fichera, S., Galluccio, L., Grancagnolo, S.C., Morabito, G., Palazzo, S., 2015. Operetta: an openflow-based remedy to mitigate tcp synflood attacks against web servers. Comput. Network. 92, 89–100.
Filsfils, C., Nainar, N.K., Pignataro, C., Cardona, J.C., Francois, P., 2015. The segment routing architecture. In: Proc. of IEEE GLOBECOM, pp. 1–6.
Fu, Y., Bi, J., Gao, K., Chen, Z., Wu, J., Hao, B., 2014. Orion: a hybrid hierarchical control plane of software-defined networking for large-scale networks. In: Proc. of IEEE ICNP, pp. 569–576.
Fu, Y., Bi, J., Chen, Z., Gao, K., Zhang, B., Chen, G., Wu, J., 2015. A hybrid hierarchical control plane for flow-based large-scale software-defined networks. IEEE Trans. Netw. Serv. Manag. 12 (2), 117–131.
Gharakheili, H.H., Sivaraman, V., Vishwanath, A., Exton, L., Matthews, J., Russell, C., 2016. Sdn apis and models for two-sided resource management in broadband access networks. IEEE Trans. Netw. Serv. Manag. 13 (4), 823–834.
Ghasemi, M., Benson, T., Rexford, J., 2017. Dapper: data plane performance diagnosis of tcp. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 61–74.
Ghorbani, S., Caesar, M., 2012. Walk the line: consistent network updates with bandwidth guarantees. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 67–72.
Ghorbani, S., Godfrey, B., 2014. Towards correct network virtualization. In: Proc. of ACM SIGCOMM Computer Communication Review, vol. 44, pp. 109–114.
Gibb, G., Zeng, H., McKeown, N., 2012. Outsourcing network functionality. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 73–78.
Giorgetti, A., Paolucci, F., Cugini, F., Castoldi, P., 2015. Dynamic restoration with GMPLS and SDN control plane in elastic optical networks. J. Opt. Commun. Netw. 7 (2), A174–A182.
Gong, Y., Wang, X., Malboubi, M., Wang, S., Xu, S., Chuah, C.-N., 2015. Towards accurate online traffic matrix estimation in software-defined networks. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–7.
Gonzlez, O., Shrikumar, H., Stankovic, J.A., Ramamritham, K., 1997. Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling. In: Proc. of IEEE Real-Time Systems Symposium, pp. 79–89.
Guck, J.W., Reisslein, M., Kellerer, W., 2016. Function split between delay-constrained routing and resource allocation for centrally managed qos in industrial networks. IEEE Trans. Ind. Inf. 12 (6), 2050–2061.
Guo, Y., Wang, Z., Yin, X., Shi, X., Wu, J., 2014. Traffic engineering in sdn/ospf hybrid network. In: Proc. of IEEE ICNP, pp. 563–568.
Guo, L., Pang, J., Walid, A., 2016. Dynamic service function chaining in sdn-enabled networks with middleboxes. In: Proc. of ICNP. IEEE, pp. 1–10.
Gupta, M., Sommers, J., Barford, P., 2013. Fast, accurate simulation for SDN prototyping. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 31–36.
Gutz, S., Story, A., Schlesinger, C., Foster, N., 2012. Splendid Isolation: a slice abstraction for software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 79–84.
Hand, R., Keller, E., 2014. ClosedFlow: OpenFlow-like control over proprietary devices. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 7–12.
Handigol, N., Heller, B., Jeyakumar, V., Mazires, D., McKeown, N., 2012. Where is the debugger for my software-defined network? In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 55–60.
Hao, F., Kodialam, M., Lakshman, T., 2016. Optimizing restoration with segment routing. In: Proc. of IEEE INFOCOM, pp. 1–9.
Hartert, R., Vissicchio, S., Schaus, P., Bonaventure, O., Filsfils, C., Telkamp, T., Francois, P., 2015. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In: Proc. of IEEE INFOCOM, pp. 15–28.
Hassas Yeganeh, S., Ganjali, Y., 2012. Kandoo: a framework for efficient and scalable offloading of control applications. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 19–24.
Heller, B., Sherwood, R., McKeown, N., 2012. The controller placement problem. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 7–12.
Heller, B., Scott, C., McKeown, N., Shenker, S., Wundsam, A., Zeng, H., Whitlock, S., Jeyakumar, V., Handigol, N., McCauley, J., et al., 2013. Leveraging SDN layering to systematically troubleshoot networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 37–42.
Honda, M., Huici, F., Lettieri, G., Rizzo, L., 2015. mswitch: a highly-scalable, modular software switch. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–13.
Hong, D.K., Ma, Y., Banerjee, S., Mao, Z.M., 2016. Incremental deployment of sdn in hybrid enterprise and isp networks. In: Proc. of ACM Symposium on SDN Research, pp. 1–7.
Hu, H., Han, W., Ahn, G.-J., Zhao, Z., 2014. FLOWGUARD: building robust firewalls for software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 97–102.
Hu, H., Wen, Y., Gao, Y., Chua, T.S., Li, X., 2015. Toward an sdn-enabled big data platform for social tv analytics. IEEE Netw. 29 (5), 43–49.
Hua, J., Ge, X., Zhong, S., 2016. Foum: a flow-ordered consistent update mechanism for software-defined networking in adversarial settings. In: Proc. of IEEE INFOCOM, pp. 1–9.

Huang, H., Guo, S., Li, P., Ye, B., Stojmenovic, I., 2015. Joint optimization of rule placement and traffic engineering for qos provisioning in software defined network. IEEE Trans. Comput. 64 (12), 3488–3499.
Huang, M., Liang, W., Xu, Z., Xu, W., Guo, S., Xu, Y., 2016a. Dynamic routing for network throughput maximization in software-defined networks. In: Proc. of IEEE INFOCOM, pp. 1–9.
Huang, J.F., Chang, G.Y., Wang, C.F., Lin, C.H., 2016b. Heterogeneous flow table distribution in software-defined networks. IEEE Trans. Emerg. Top. Comput. 4 (2), 252–261.
Huang, H., Guo, S., Li, P., Liang, W., Zomaya, A.Y., 2016c. Cost minimization for rule caching in software defined networking. IEEE Trans. Parallel Distrib. Syst. 27 (4), 1007–1016.
Jafarian, J.H., Al-Shaer, E., Duan, Q., 2012. OpenFlow random host mutation: transparent moving target defense using SDN. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 127–132.
Jeong, C., Ha, T., Narantuya, J., Lim, H., Kim, J., 2014. Scalable network intrusion detection on virtual SDN environment. In: Proc. of IEEE International Conference on CloudNet, pp. 264–265.
Ji, Y., Zhang, J., Zhao, Y., Li, H., Yang, Q., Ge, C., Xiong, Q., Xue, D., Yu, J., Qiu, S., 2014. All optical switching networks with energy-efficient technologies from components level to network level. IEEE J. Sel. Area. Commun. 32 (8), 1600–1614.
Jin, X., Farrington, N., Rexford, J., 2016. Your data center switch is trying too hard. In: Proc. of ACM Symposium on SDN Research, pp. 1–6.
Jin, C., Lumezanu, C., Xu, Q., Mekky, H., Zhang, Z.-L., Jiang, G., 2017. Magneto: unified fine-grained path control in legacy and openflow hybrid networks. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 75–87.
Jo, J., Lee, S., Kim, J.W., 2014. Software-defined home networking devices for multi-home visual sharing. IEEE Trans. Consum. Electron. 60 (3), 534–539.
Jyothi, S.A., Dong, M., Godfrey, P., 2015. Towards a flexible data center fabric with source routing. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–8.
Kampanakis, P., Perros, H., Beyene, T., 2014. Sdn-based solutions for moving target defense network protection. In: Proc. of IEEE International Symposium on WoWMoM, pp. 1–6.
Kang, N., Liu, Z., Rexford, J., Walker, D., 2013. Optimizing the one big switch abstraction in software-defined networks. In: Proc. of ACM ENET, pp. 13–24.
Kar, B., Wu, E.H.K., Lin, Y.D., 2016. The budgeted maximum coverage problem in partially deployed software defined networks. IEEE Trans. Netw. Serv. Manag. 13 (3), 394–406.
Katta, N.P., Rexford, J., Walker, D., 2013. Incremental consistent updates. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 49–54.
Katta, N., Zhang, H., Freedman, M., Rexford, J., 2015. Ravana: controller fault-tolerance in software-defined networking. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–12.
Khurshid, A., Zou, X., Zhou, W., Caesar, M., Godfrey, P.B., 2013. Veriflow: verifying network-wide invariants in real time. In: Proc. of the USENIX Symposium on Networked Systems Design and Implementation, pp. 15–27.
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: Proc. of OSDI, vol. 10, pp. 1–6.
Kozat, U.C., Liang, G., Kokten, K., 2014. On diagnosis of forwarding plane via static forwarding rules in software defined networks. In: Proc. of IEEE INFOCOM, pp. 1716–1724.
Kreutz, D., Ramos, F., Verissimo, P., 2013. Towards secure and dependable software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 55–60.
Kreutz, D., Ramos, F.M.V., Verssimo, 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: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 133–138.
Kuang, L., Yang, L.T., Qiu, K., 2016. Tensor-based software-defined internet of things. IEEE Wireless Commun. 23 (5), 84–89.
Kuerban, M., Tian, Y., Yang, Q., Jia, Y., Huebert, B., Poss, D., 2016. FlowSec: DOS attack mitigation strategy on SDN controller. In: Proc. of IEEE NAS, pp. 1–2.
Kuzniar, M., Peresini, P., Kosti, D., 2014. Providing reliable fib update acknowledgments in sdn. In: Proc. of ACM International on Conference on Emerging Networking Experiments and Technologies, pp. 415–422.
Lai, C. f., Hwang, R. h., Chao, H. c., Hassan, M.M., Alamri, A., 2015. A buffer-aware http live streaming approach for sdn-enabled 5g wireless networks. IEEE Netw. 29 (1), 49–55.
Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21 (7), 558–565.
Lange, S., et al., 2015. Heuristic approaches to the controller placement problem in large scale SDN networks. IEEE Trans. Netw. Serv. Manag. 12 (1), 4–17.
Lara, A., Ramamurthy, B., 2016. Opensec: policy-based security using software-defined networking. IEEE Trans. Netw. Serv. Manag. 13 (1), 30–42.
Laufer, R., Gallo, M., Perino, D., Nandugudi, A., 2016. Climb: enabling network function composition with click middleboxes. ACM SIGCOMM Comput. Commun. Rev. 46 (4), 17–22.
Lee, B., Park, S.H., Shin, J., Yang, S., 2014. Iris: the openflow-based recursive sdn controller. In: Proc. of IEEE ACT, pp. 1227–1231.
Lvai, T., Pelle, I., Nmeth, F., Gulys, A., 2015. Epoxide: a modular prototype for sdn troubleshooting. In: Proc. of ACM SIGCOMM, pp. 359–360.
Levin, D., Wundsam, A., Heller, B., Handigol, N., Feldmann, A., 2012. Logically centralized?: state distribution trade-offs in SDNs. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 1–6.
Li, H., Li, P., Guo, S., Nayak, A., 2014a. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. IEEE Trans. Cloud Comput. 2 (4), 436–447.
Li, J., Hyun, J., Yoo, J.-H., Baik, S., Hong, J.W.-K., 2014b. Scalable failover method for data center networks using openflow. In: Proc. of IEEE NOMS, pp. 1–6.
Li, D., Shang, Y., Chen, C., 2014c. Software defined green data center network with exclusive routing. In: Proc. of IEEE INFOCOM, pp. 1743–1751.
Li, Y., Zheng, F., Chen, M., Jin, D., 2015a. A unified control and optimization framework for dynamical service chaining in software-defined nfv system. IEEE Wireless Commun. 22 (6), 15–23.
Li, D., Shang, Y., He, W., Chen, C., 2015b. Exr: greening data center network with software defined exclusive routing. IEEE Trans. Comput. 64 (9), 2534–2544.
Li, L., Chou, W., Zhou, W., Luo, M., 2016a. Design patterns and extensibility of rest api for networking applications. IEEE Trans. Netw. Serv. Manag. 13 (1), 154–167.
Li, H., Dong, M., Ota, K., 2016b. Control plane optimization in software-defined vehicular ad hoc networks. IEEE Trans. Veh. Technol. 65 (10), 7895–7904.
Li, T., Zhou, H., Luo, H., Yu, S., 2018. Service: a software defined framework for integrated space-terrestrial satellite communication. IEEE Trans. Mob. Comput. 17 (3), 703–716.
Lin, P., Bi, J., Hu, H., 2012. VCP: a virtualization cloud platform for SDN intra-domain production network. In: Proc. of IEEE ICNP, pp. 1–2.
Lin, H., Chen, C., Wang, J., Qi, J., Jin, D., Kalbarczyk, Z., Iyer, R.K., 2018. Self-healing attack-resilient pmu network for power system operation. IEEE Trans. Smart Grid 9 (3), 1551–1565.
Liu, X., Shirazipour, M., Yu, M., Zhang, Y., 2016a. Mozart: temporal coordination of measurement. In: Proc. of ACM Symposium on SDN Research, pp. 1–12.
Liu, K., Ng, J.K.Y., Lee, V.C.S., Son, S.H., Stojmenovic, I., 2016b. Cooperative data scheduling in hybrid vehicular ad hoc networks: vanet as a software defined network. IEEE/ACM Trans. Netw. 24 (3), 1759–1773.
Liu, J., Lai, Y., Zhang, S., 2017. Fl-guard: a detection and defense system for ddos attack in sdn. In: Proc. of IEEE International Conference on Cryptography, Security and Privacy, pp. 107–111.
Liu, C.F., Samarakoon, S., Bennis, M., Poor, H.V., 2018. Fronthaul-aware software-defined wireless networks: resource allocation and user scheduling. IEEE Trans. Wirel. Commun. 17 (1), 533–547.
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.
Lu, G., Miao, R., Xiong, Y., Guo, C., 2012. Using CPU as a traffic co-processing unit in commodity switches. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 31–36.
Luo, S., Wu, J., Li, J., Guo, L., 2016. A multi-stage attack mitigation mechanism for software-defined home networks. IEEE Trans. Consum. Electron. 62 (2), 200–207.
Maksymyuk, T., Kyryk, M., Jo, M., 2016. Comprehensive spectrum management for heterogeneous networks in lte-u. IEEE Wireless Commun. 23 (6), 8–15.
Martinello, M., Ribeiro, M.R.N., de Oliveira, R.E.Z., de Angelis Vitoi, R., 2014. Keyflow: a prototype for evolving sdn toward core network fabrics. IEEE Netw. 28 (2), 12–19.
Martini, B., Paganelli, F., Mohammed, A., Gharbaoui, M., Sgambelluri, A., Castoldi, P., 2015. Sdn controller for context-aware data delivery in dynamic service chaining. In: Proc. of IEEE NetSoft, pp. 1–5.
Martins, J., Ahmed, M., Raiciu, C., Huici, F., 2013. Enabling fast, dynamic network processing with clickos. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 67–72.
Matsumoto, S., Hitz, S., Perrig, A., 2014. Fleet: defending SDNs from malicious administrators. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 103–108.
McGeer, R., 2012. A safe, efficient update protocol for OpenFlow networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 61–66.
Mekky, H., Hao, F., Mukherjee, S., Zhang, Z.-L., Lakshman, T., 2014. Application-aware data plane processing in SDN. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 13–18.
Melis, L., Asghar, H.J., De Cristofaro, E., Kaafar, M.A., 2016. Private processing of outsourced network functions: feasibility and constructions. In: Proc. of ACM International Workshop on Security in SDN-NFV, pp. 39–44.
Miserez, J., Bielik, P., El-Hassany, A., Vanbever, L., Vechev, M., 2015. Sdnracer: detecting concurrency violations in software-defined networks. In: Proc. of ACM SIGCOMM Symposium on Software Defined Networking Research, p. 22.
Mizrahi, T., Moses, Y., 2016. Time4: time for sdn. IEEE Trans. Netw. Serv. Manag. 13 (3), 433–446.
Mizrahi, T., Saat, E., Moses, Y., 2016. Timed consistent network updates in software-defined networks. IEEE/ACM Trans. Netw. 24 (6), 3412–3425.
Monsanto, C., Reich, J., Foster, N., Rexford, J., Walker, D., et al., 2013. Composing software defined networks. In: NSDI, vol. 13, pp. 1–13.
Moshref, M., Yu, M., Govindan, R., 2013. Resource/accuracy tradeoffs in software-defined measurement. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 73–78.
Moshref, M., Bhargava, A., Gupta, A., Yu, M., Govindan, R., 2014. Flow level state transition as a new switch primitive for SDN. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 61–66.
Moshref, M., Yu, M., Govindan, R., Vahdat, A., 2015. Dream: dynamic resource allocation for software-defined measurement. ACM SIGCOMM Comput. Commun. Rev. 44 (4), 419–430.

Mu, M., Broadbent, M., Farshad, A., Hart, N., Hutchison, D., Ni, Q., Race, N., 2016. A scalable user fairness model for adaptive video streaming over sdn-assisted future networks. IEEE J. Sel. Area. Commun. 34 (8), 2168–2184.
Najjar, W., Gaudiot, J.L., 1990. Network resilience: a measure of network fault tolerance. IEEE Trans. Comput. 39 (2), 174–181.
Nelson, T., Ferguson, A.D., Scheer, M.J., Krishnamurthi, S., 2014. Tierless programming and reasoning for software-defined networks. In: Proc. of USENIX Symposium on NSDI, pp. 519–531.
Nelson, T., Ferguson, A.D., Yu, D., Fonseca, R., Krishnamurthi, S., 2015. Exodus: toward automatic migration of enterprise network configurations to SDNs. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–7.
Newport, C., Zhou, W., 2015. The (surprising) computational power of the sdn data plane. In: Proc. of IEEE INFOCOM, pp. 496–504.
Nguyen, X.-N., Saucez, D., Barakat, C., Turletti, T., 2014. Optimizing rules placement in OpenFlow networks: trading routing for better efficiency. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 127–132.
Othman, O.M., Okamura, K., 2013. Securing distributed control of software defined networks. Int. J. Comput. Sci. Netw. Secur. 13 (9), 5.
Pan, H., Guan, H., Liu, J., Ding, W., Lin, C., Xie, G., 2013. The FlowAdapter: enable flexible multi-table processing on legacy hardware. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 85–90.
Pan, H., Li, Z., Xie, G., He, P., Guan, H., Mathy, L., 2018. Efficient action computation for compositional sdn policies. IEEE Trans. Netw. Serv. Manag. 15 (1), 387–401.
Paolucci, F., 2018. Network service chaining using segment routing in multi-layer networks. IEEE/OSA J. Opt. Commun. Netw. 10 (6), 582–592.
Paolucci, F., Castro, A., Cugini, F., Velasco, L., Castoldi, P., 2014. Multipath restoration and bitrate squeezing in sdn-based elastic optical networks. Photonic Netw. Commun. 28 (1), 45–57.
Paolucci, F., Civerchia, F., Sgambelluri, A., Giorgetti, A., Cugini, F., Castoldi, P., 2019. P4 edge node enabling stateful traffic engineering and cyber security. J. Opt. Commun. Netw. 11 (1), A84–A95.
Paris, S., Destounis, A., Maggi, L., Paschos, G.S., Leguay, J., 2016. Controlling flow reconfigurations in sdn. In: Proc. of IEEE INFOCOM, pp. 1–9.
Pashkov, V., Shalimov, A., Smeliansky, R., 2014. Controller failover for sdn enterprise networks. In: Proc. of IEEE MoNeTeC, pp. 1–6.
Pedreno-Manresa, J.J., Khodashenas, P.S., Siddiqui, M.S., Pavon-Marino, P., 2018. On the need of joint bandwidth and nfv resource orchestration: a realistic 5g access network use case. IEEE Commun. Lett. 22 (1), 145–148.
Pereni, P., Kuzniar, M., Canini, M., Kosti, D., 2014. ESPRES: transparent SDN update scheduling. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 73–78.
Pereni, P., Kuniar, M., Kosti, D., 2015. Monocle: dynamic, fine-grained data plane monitoring. In: Proc. of ACM Emerging Networking Experiments and Technologies, p. 32.
Pereni, P., Kuniar, M., Kosti, D., 2018. Dynamic, fine-grained data plane monitoring with monocle. IEEE/ACM Trans. Netw. 26 (1), 534–547.
Phemius, K., Bouet, M., Leguay, J., 2014. Disco: distributed multi-domain sdn controllers. In: Proc. of IEEE NOMS, pp. 1–4.
Polverini, M., Baiocchi, A., Cianfrani, A., Iacovazzi, A., Listanti, M., 2016. The power of sdn to improve the estimation of the isp traffic matrix through the flow spread concept. IEEE J. Sel. Area. Commun. 34 (6), 1904–1913.
Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., Gu, G., 2012. A security enforcement kernel for OpenFlow networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 121–126.
Prakash, C., Lee, J., Turner, Y., Kang, J.-M., Akella, A., Banerjee, S., Clark, C., Ma, Y., Sharma, P., Zhang, Y., 2015. Pga: Using Graphs to Express and Automatically Reconcile Network Policies, vol. 45, pp. 29–42.
Qazi, Z.A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., Yu, M., 2013. Simple-fying middlebox policy enforcement using sdn. In: Proc. of ACM SIGCOMM, pp. 27–38.
Rahman, M.M., Despins, C., Affes, S., 2016. Design optimization of wireless access virtualization based on cost qos trade-off utility maximization. IEEE Trans. Wirel. Commun. 15 (9), 6146–6162.
Reitblatt, M., Canini, M., Guha, A., Foster, N., 2013. FatTire: declarative fault tolerance for software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 109–114.
Rifai, M., Huin, N., Caillouet, C., Giroire, F., Lopez-Pacheco, D., Moulierac, J., Urvoy-Keller, G., 2015. Too many sdn rules? compress them with minnie. In: Proc. of IEEE GLOBECOM, pp. 1–7.
Riggio, R., Marina, M.K., Schulz-Zander, J., Kuklinski, S., Rasheed, T., 2015. Programming abstractions for software-defined wireless networks. IEEE Trans. Netw. Serv. Manag. 12 (2), 146–162.
Ros, F.J., Ruiz, P.M., 2014. Five nines of southbound reliability in software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 31–36.
Rckert, J., Blendin, J., Hark, R., Hausheer, D., 2016. Flexible, efficient, and scalable software-defined over-the-top multicast for isp environments with dynsdm. IEEE Trans. Netw. Serv. Manag. 13 (4), 754–767.
Sallahi, A., St-Hilaire, M., 2017. Expansion model for the controller placement problem in software defined networks. IEEE Commun. Lett. 21 (2), 274–277.
Salsano, S., Ventre, P.L., Lombardo, F., Siracusano, G., Gerola, M., Salvadori, E., Santuari, M., Campanella, M., Prete, L., 2016. Hybrid ip/sdn networking: open implementation and experiment management tools. IEEE Trans. Netw. Serv. Manag. 13 (1), 138–153.
Sambo, N., Meloni, G., Paolucci, F., Cugini, F., Secondini, M., Fresi, F., Pot, L., Castoldi, P., 2014. Programmable transponder, code and differentiated filter configuration in elastic optical networks. J. Light. Technol. 32 (11), 2079–2086.
Saunders, R., Cho, J., Banerjee, A., Rocha, F., Van der Merwe, J., 2016. P2p offloading in mobile networks using sdn. In: Proc. of ACM Symposium on SDN Research, pp. 1–7.
Schmid, S., Suomela, J., 2013. Exploiting locality in distributed SDN control. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 121–126.
Scott-Hayward, S., Kane, C., Sezer, S., 2014. Operationcheckpoint: sdn application control. In: Proc. of IEEE ICNP, pp. 618–623.
Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., Rao, N., 2013. Are we ready for sdn? implementation challenges for software-defined networks. IEEE Commun. Mag. 51 (7), 36–43.
Sgambelluri, A., Paolucci, F., Cugini, F., Valcarenghi, L., Castoldi, P., 2013a. Generalized sdn control for access/metro/core integration in the framework of the interface to the routing system (i2rs). In: Proc. of IEEE GLOBECOM Workshops, pp. 1216–1220.
Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., Castoldi, P., 2013b. Openflow-based segment protection in ethernet networks. J. Opt. Commun. Netw. 5 (9), 1066–1075.
Sgambelluri, A., Paolucci, F., Giorgetti, A., Cugini, F., Castoldi, P., 2016. Experimental demonstration of segment routing. J. Light. Technol. 34 (1), 205–212.
Sieber, C., Blenk, A., Basta, A., Hock, D., Kellerer, W., 2016. Towards a programmable management plane for sdn and legacy networks. In: Proc. of IEEE NetSoft, pp. 319–327.
Sinha, Y., Bhatia, S., Shekhawat, V.S., 2017. Mpls based hybridization in sdn. In: Proc. IEEE SDS, pp. 156–161.
Song, H., 2013. Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 127–132.
Song, S., Park, H., Choi, B.Y., Choi, T., Zhu, H., 2017. Control path management framework for enhancing software-defined network (sdn) reliability. IEEE Trans. Netw. Serv. Manag. 14 (2), 302–316.
Soni, H., Dabbous, W., Turletti, T., Asaeda, H., 2017. Nfv-based scalable guaranteed-bandwidth multicast service for software defined isp networks. IEEE Trans. Netw. Serv. Manag. 14 (4), 1157–1170.
Sood, K., Yu, S., Xiang, Y., 2016. Performance analysis of software-defined network switch using *mgeo*1 model. IEEE Commun. Lett. 20 (12), 2522–2525.
Soul, R., Basu, S., Kleinberg, R., Sirer, E.G., Foster, N., 2014. Merlin: programming the big switch. In: The Open Networking Summit.
Stephens, B., Cox, A.L., Rixner, S., 2016. Scalable multi-failure fast failover via forwarding table compression. In: Proc. of ACM Symposium on SDN Research, pp. 1–12.
Suh, D., Pack, S., 2018. Low-complexity master controller assignment in distributed sdn controller environments. IEEE Commun. Lett. 22 (3), 490–493.
Sun, P., Vanbever, L., Rexford, J., 2015a. Scalable programmable inbound traffic engineering. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–7.
Sun, X.S., Agarwal, A., Ng, T.S.E., 2015b. Controlling race conditions in openflow to accelerate application verification and packet forwarding. IEEE Trans. Netw. Serv. Manag. 12 (2), 263–277.
Sun, S., Kadoch, M., Gong, L., Rong, B., 2015c. Integrating network function virtualization with sdr and sdn for 4g/5g networks. IEEE Netw. 29 (3), 54–59.
Sun, P., Wen, Y., Duong, T.N.B., Xie, H., 2018. Metaflow: a scalable metadata lookup service for distributed file systems in data centers. IEEE Trans. Big Data 4 (2), 203–216.
Syed, A., Van der Merwe, J., 2016. Proteus: a network service control platform for service evolution in a mobile software defined infrastructure. In: Proc. of ACM International Conference on Mobile Computing and Networking, pp. 257–270.
Szabo, R., Kind, M., Westphal, F.J., Woesner, H., Jocha, D., Csaszar, A., 2015. Elastic network functions: opportunities and challenges. IEEE Netw. 29 (3), 15–21.
Tam, A.S.-W., Xi, K., Chao, H.J., 2011. Use of devolved controllers in data center networks. In: Proc. of IEEE INFOCOM WKSHPS, pp. 596–601.
Tan, W., Zhang, J., Peng, C., Xia, B., Kou, Y., 2014. Sdn-enabled converged networks. IEEE Wireless Commun. 21 (6), 79–85.
Tanha, M., Sajjadi, D., Ruby, R., Pan, J., 2018. Traffic engineering enhancement by progressive migration to SDN. IEEE Commun. Lett. 22 (3), 438–441.
Taylor, C.R., MacFarland, D.C., Smestad, D.R., Shue, C.A., 2016. Contextual, flow-based access control with scalable host-based SDN techniques. In: Proc. of IEEE INFOCOM, pp. 1–9.
Thyagaturu, A.S., Dashti, Y., Reisslein, M., 2016. Sdn-based smart gateways (sm-gws) for multi-operator small cell network management. IEEE Trans. Netw. Serv. Manag. 13 (4), 740–753.
Tootoonchian, A., Ganjali, Y., 2010. Hyperflow: a distributed control plane for openflow. In: Proc. of the Internet Network Management Conference on Enterprise Networking, pp. 1–6.
Trevisan, M., Drago, I., Mellia, M., Song, H.H., Baldi, M., 2018. Awesome: big data for automatic web service management in sdn. IEEE Trans. Netw. Serv. Manag. 15 (1), 13–26.
Tuncer, D., Charalambides, M., Clayman, S., Pavlou, G., 2015. Adaptive resource management and control in software defined networks. IEEE Trans. Netw. Serv. Manag. 12 (1), 18–33.
Van Adrichem, N.L., Van Asten, B.J., Kuipers, F.A., 2014. Fast recovery in software-defined networks. In: Proc. of IEEE WSDN, pp. 61–66.
Vanbever, L., Reich, J., Benson, T., Foster, N., Rexford, J., 2013. HotSwap: correct and efficient controller upgrades for software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 133–138.
Varadharajan, V., Karmakar, K., Tupakula, U., Hitchens, M., 2019. A policy-based security architecture for software-defined networks. IEEE Trans. Inf. Forensics Secur. 14 (4), 897–912.

Varvello, M., Laufer, R., Zhang, F., Lakshman, T.V., 2016. Multilayer packet classification with graphics processing units. IEEE/ACM Trans. Netw. 24 (5), 2728–2741.
Velasco, L., Gifre, L., Izquierdo-Zaragoza, J.-L., Paolucci, F., Vela, A.P., Sgambelluri, A., Ruiz, M., Cugini, F., 2018. An architecture to support autonomic slice networking. J. Light. Technol. 36 (1), 135–141.
Vissicchio, S., Tilmans, O., Vanbever, L., Rexford, J., 2015. Central control over distributed routing. ACM SIGCOMM Comput. Commun. Rev. 45 (4), 43–56.
Vrs, P., Kiss, A., 2016. Security middleware programming using p4. In: Proc. of Human Aspects of Information Security, Privacy, and Trust, pp. 277–287.
Wang, S., Huang, X., 2016. Aggregation points planning for software-defined network based smart grid communications. In: Proc. of IEEE INFOCOM, pp. 1–9.
Wang, G., Ng, T., Shaikh, A., 2012. Programming your network at run-time for big data applications. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 103–108.
Wang, J., Wang, Y., Hu, H., Sun, Q., Shi, H., Zeng, L., 2013. Towards a security-enhanced firewall application for openflow networks. In: Proc. of Springer International Symposium Cyberspace Safety and Security, pp. 92–103.
Wang, H., Li, Y., Jin, D., Hui, P., Wu, J., 2016a. Saving energy in partially deployed software defined networks. IEEE Trans. Comput. 65 (5), 1578–1592.
Wang, K., Li, H., Yu, F.R., Wei, W., 2016b. Virtual resource allocation in software-defined information-centric cellular networks with device-to-device communications and imperfect csi. IEEE Trans. Veh. Technol. 65 (12), 10011–10021.
Wang, T., Liu, F., Guo, J., Xu, H., 2016c. Dynamic sdn controller assignment in data center networks: stable matching with transfers. In: Proc. of IEEE INFOCOM, pp. 1–9.
Wang, J.M., Wang, Y., Dai, X., Bensaou, B., 2016d. Sdn-based multi-class qos guarantee in inter-data center communications. IEEE Trans. Cloud Comput. PP (99), 1.
Wang, W., He, W., Su, J., 2017a. Boosting the benefits of hybrid sdn. In: Proc. of IEEE ICDCS, pp. 2165–2170.
Wang, P., Xu, H., Huang, L., He, J., Meng, Z., 2017b. Control link load balancing and low delay route deployment for software defined networks. IEEE J. Sel. Area. Commun. 35 (11), 2446–2456.
Wang, X., Wang, C., Zhang, J., Zhou, M., Jiang, C., 2017c. Improved rule installation for real-time query service in software-defined internet of vehicles. IEEE Trans. Intell. Transp. Syst. 18 (2), 225–235.
Wang, K., Yang, K., Chen, H.H., Zhang, L., 2017d. Computation diversity in emerging networking paradigms. IEEE Wireless Commun. 24 (1), 88–94.
Wang, H., Li, Y., Zhang, Y., Jin, D., 2017e. Virtual machine migration planning in software-defined networks. IEEE Trans. Cloud Comput. 1.
Wang, P., Xu, H., Huang, L., Qian, C., Wang, S., Sun, Y., 2018. Minimizing controller response time through flow redirecting in SDNs. IEEE/ACM Trans. Netw. 26 (1), 562–575.
Wei, L., Zhang, C., Gong, Y., Fang, Y., Chen, K., 2016. A firewall of two clouds: preserving outsourced firewall policy confidentiality with heterogeneity. In: Proc. of IEEE GLOBECOM, pp. 1–6.
Wen, X., Chen, Y., Hu, C., Shi, C., Wang, Y., 2013. Towards a secure controller platform for openflow applications. In: Proc. of ACM SIGCOMM Workshop on HotSDN, pp. 171–172.
Wen, X., Diao, C., Zhao, X., Chen, Y., Li, L.E., Yang, B., Bu, K., 2014. Compiling minimum incremental update for modular SDN languages. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 193–198.
Wen, X., Bu, K., Yang, B., Chen, Y., Li, L.E., Chen, X., Yang, J., Leng, X., 2017. Rulescope: inspecting forwarding faults for software-defined networking. IEEE/ACM Trans. Netw. 2347–2360.
Williams, D., Jamjoom, H., 2013. Cementing high availability in openflow with rulebricks. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 139–144.
Wood, T., Ramakrishnan, K.K., Hwang, J., Liu, G., Zhang, W., 2015. Toward a software-based network: integrating software defined networking and network function virtualization. IEEE Netw. 29 (3), 36–41.
Wu, J., Liu, D., Huang, X.L., Luo, C., Cui, H., Wu, F., 2015. Dac-ran: a data-assisted cloud radio access network for visual communications. IEEE Wireless Commun. 22 (3), 130–136.
Wu, Y., Zhang, Z., Wu, C., Guo, C., Li, Z., Lau, F.C.M., 2017. Orchestrating bulk data transfers across geo-distributed datacenters. IEEE Trans. Cloud Comput. 5 (1), 112–125.
Wu, J., Dong, M., Ota, K., Li, J., Guan, Z., 2018. Big data analysis-based secure cluster management for optimized control plane in software-defined networks. IEEE Trans. Netw. Serv. Manag. 15 (1), 27–38.
Wundsam, A., Levin, D., Seetharaman, S., Feldmann, A., et al., 2011. Ofrewind: enabling record and replay troubleshooting for networks. In: Proc. of USENIX Annual Technical Conference, pp. 15–17.
Xia, W., Wen, Y., Foh, C.H., Niyato, D., Xie, H., 2015. A survey on software-defined networking. IEEE Commun. Surv. Tutor. 17 (1), 27–51.
Xie, K., Huang, X., Hao, S., Ma, M., Zhang, P., Hu, D., 2016. Mc: improving energy efficiency via elastic multi-controller sdn in data center networks. IEEE Access 4, 6780–6791.
Xie, J., Guo, D., Zhu, X., Ren, B., Chen, H., 2017. Minimal fault-tolerant coverage of controllers in iaas datacenters. IEEE Trans. Serv. Comput. 1.
Xing, C., Ding, K., Hu, C., Chen, M., 2016. Sample and fetch-based large flow detection mechanism in software defined networks. IEEE Commun. Lett. 20 (9), 1764–1767.
Xiong, Y., Li, Y., Zhou, B., Wang, R., Rouskas, G.N., 2018. Sdn enabled restoration with triggered precomputation in elastic optical inter-datacenter networks. J. Opt. Commun. Netw. 10 (1), 24–34.
Xu, Y., Liu, Y., Singh, R., Tao, S., 2015. Identifying sdn state inconsistency in openstack. In: Proc. of ACM SIGCOMM Symposium on SDN Research, pp. 1–11.
Xu, Y., Liu, Y., Singh, R., Tao, S., 2016. Sdn state inconsistency verification in openstack. Comput. Network. 110, 364–376.
Xu, H., Li, X.-Y., Huang, L., Deng, H., Huang, H., Wang, H., 2017a. Incremental deployment and throughput maximization routing for a hybrid sdn. IEEE/ACM Trans. Netw. 1861–1875.
Xu, H., Yu, Z., Qian, C., Li, X.Y., Liu, Z., Huang, L., 2017b. Minimizing flow statistics collection cost using wildcard-based requests in sdns. IEEE/ACM Trans. Netw. 25 (6), 3587–3601.
Xu, H., Huang, H., Chen, S., Zhao, G., Huang, L., 2018. Achieving high scalability through hybrid switching in software-defined networking. IEEE/ACM Trans. Netw. 26 (1), 618–632.
Xue, N., Chen, X., Gong, L., Li, S., Hu, D., Zhu, Z., 2015. Demonstration of openflow-controlled network orchestration for adaptive svc video manycast. IEEE Trans. Multimed. 17 (9), 1617–1629.
Yaghoubi, F., Furdek, M., Rostami, A., hln, P., Wosinska, L., 2018. Consistency-aware weather disruption-tolerant routing in sdn-based wireless mesh networks. IEEE Trans. Netw. Serv. Manag. 15 (2), 582–595.
Yan, B., Xu, Y., Xing, H., Xi, K., Chao, H.J., 2014. CAB: a reactive wildcard rule caching system for software-defined networks. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 163–168.
Yang, J., Zhu, K., Ran, Y., Cai, W., Yang, E., 2017. Jointadmissioncontrolandroutingviaapproximate dynamic programming for streaming video over software-defined networking. IEEE Trans. Multimed. 19 (3), 619–631.
Yao, J., Wang, Z., Yin, X., Shiyz, X., Wu, J., 2014. Formal modeling and systematic black-box testing of sdn data plane. In: Proc. of IEEE ICNP, pp. 179–190.
Yazici, V., Sunay, M.O., Ercan, A.O., 2014. Controlling a Software-Defined Network via Distributed Controllers. arXiv:1401.7651.
Yoshida, Y., Maruta, A., Kitayama, K.-i., Nishihara, M., Tanaka, T., Takahara, T., Rasmussen, J.C., Yoshikane, N., Tsuritani, T., Morita, I., et al., 2015. Sdn-based network orchestration of variable-capacity optical packet switching network over programmable flexi-grid elastic optical path network. J. Light. Technol. 33 (3), 609–617.
Yousaf, F.Z., Bredel, M., Schaller, S., Schneider, F., 2017. Nfv and sdn-key technology enablers for 5g networks. IEEE J. Sel. Area. Commun. 35 (11), 2468–2478.
Yu, Y., Qian, C., Li, X., 2014. Distributed and collaborative traffic monitoring in SDNs. In: Proc. of ACM SIGCOMM Workshop on HOTSDN, pp. 85–90.
Zamani, A.R., Zou, M., Diaz-Montes, J., Petri, I., Rana, O., Anjum, A., Parashar, M., 2017. Deadline constrained video analysis via in-transit computational environments. IEEE Trans. Serv. Comput. 1.
Zhang, Y., Beheshti, N., Beliveau, L., Lefebvre, G., Manghirmalani, R., Mishra, R., Patneyt, R., Shirazipour, M., Subrahmaniam, R., Truchan, C., et al., 2013a. Steering: a software-defined networking for inline service chaining. In: Proc. of IEEE ICNP, pp. 1–10.
Zhang, J., Zhang, J., Zhao, Y., Yang, H., Yu, X., Wang, L., Fu, X., 2013b. Experimental demonstration of openflow-based control plane for elastic lightpath provisioning in flexi-grid optical networks. Optic Express 21 (2), 1364–1373.
Zhang, P., Zhao, Y., Wang, Y., Jin, Y., 2016. Enhancing network performance tomography in software-defined cloud network. IEEE Commun. Lett. 1.
Zhang, X., Cheng, Z., Lin, R., He, L., Yu, S., Luo, H., 2017. Local fast reroute with flow aggregation in software defined networks. IEEE Commun. Lett. 21 (4), 785–788.
Zhao, S., Medhi, D., 2017. Application-aware network design for hadoop mapreduce optimization using software-defined networking. IEEE Trans. Netw. Serv. Manag. 14 (4), 804–816.
Zheng, J., Chen, G., Schmid, S., Dai, H., Wu, J., Ni, Q., 2017. Scheduling congestion- and loop-free network update in timed sdns. IEEE J. Sel. Area. Commun. 35 (11), 2542–2552.
Zhou, H., Wu, C., Cheng, Q., Liu, Q., 2017. Sdn-liru: a lossless and seamless method for sdn inter-domain route updates. IEEE/ACM Trans. Netw. 25 (4), 2473–2483.
Zhu, Y., Yan, F., Zhang, Y., Zhang, R., Shen, L., 2017. Sdn-based anchor scheduling scheme for localization in heterogeneous wsns. IEEE Commun. Lett. 21 (5), 1127–1130.

Surbhi Saraswat received M.Tech. degree in Information Technology (Wireless Communication Engineering) from Indian Institute of Information Technology (IIIT) Allahabad, India. Presently, she is pursuing her Ph.D. in the Department of Computer Science & Engineering, Indian Institute of Technology (BHU) Varanasi, India. Her research interests include fog computing, wireless ad hoc networks, and software defined networks.



Vishal Agarwal received Integrated Dual Degree (IDD) in Computer Science & Engineering from Indian Institute of Technology (BHU) Varanasi, India. He was Training and Placement Representative in year 2017 in IIT (BHU) Varanasi. His research interests include wireless sensor networks, software defined network, and fog computing.

Hari Prabhat Gupta received the B.E. degree in Computer Engineering from Government Engineering College Ajmer, Ajmer, India, the M.Tech. and Ph.D. degrees in Computer Science and Engineering from the Indian Institute of Technology Guwahati (IITG), Guwahati, India. He is currently working as Assistant Professor in Department of Computer Science and Engineering, Indian Institute of Technology (BHU), Varanasi, India. He has received a research fellowship from TATA Consultancy Services, India. His research interests include wireless sensor networks, wireless ad hoc networks, and distributed algorithms.

Rahul Mishra received the B.Tech. degree in Computer Science & Engineering from Uttar Pradesh Technical University, India. He had completed his M.Tech. from Madan Mohan Malaviya Univesity of Technology (MMMUT) Uttar Pradesh. Presently, he is pursuing his Ph.D. in the Department of Computer Science & Engineering, Indian Institute of Technology (BHU) Varanasi, India. His research interests include wireless sensor networks, wireless ad hoc networks, and algorithms.

Ashish Gupta received the M.Tech. degree in Computer Engineering from Sardar Vallabhbhai National Institute of Technology (SVNIT), Surat, India. Presently, he is pursuing his Ph.D. in the Department of Computer Science & Engineering, Indian Institute of Technology (BHU) Varanasi, India. His research interests include wireless sensor networks, data structures, and data mining.

Tanima Dutta received the Ph.D. degree in computer science and engineering from Indian Institute of Technology Guwahati (IITG), Guwahati, India, in 2014. She worked with TCS Innovation Labs, Bangalore, India. She has received TCS research fellowship and SAIL fellowship for pursuing the Ph.D. and B.Tech. degrees, respectively. She is currently working as Assistant Professor in Department of Computer Science and Engineering, Indian Institute of Technology (BHU), Varanasi, India. Her research interests include multimedia, digital forensics, wireless sensor networks, and algorithms.