Computer Networks 71 (2014) 1–30

Contents lists available at ScienceDirect

Computer Networks

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

A roadmap for traffic engineering in SDN-OpenFlow networks

Ian F. Akyildiz [a], Ahyoung Lee [a,*], Pu Wang [b], Min Luo [c], Wu Chou [c]

[a] Broadband Wireless Networking Lab, School of Electrical & Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA
[b] Department of Electrical Engineering and Computer Science, Wichita State University, Wichita, KS 67260, USA
[c] Shannon Lab, Huawei Technologies Co., Ltd., Santa Clara, USA

ARTICLE INFO

Article history:
Available online 19 June 2014

Keywords:
Software-defined networking
OpenFlow
Traffic engineering
Traffic management
Traffic analysis

ABSTRACT

Software Defined Networking (SDN) is an emerging networking paradigm that separates the network control plane from the data forwarding plane with the promise to dramatically improve network resource utilization, simplify network management, reduce operating cost, and promote innovation and evolution. Although traffic engineering techniques have been widely exploited in the past and current data networks, such as ATM networks and IP/MPLS networks, to optimize the performance of communication networks by dynamically analyzing, predicting, and regulating the behavior of the transmitted data, the unique features of SDN require new traffic engineering techniques that exploit the global network view, status, and flow patterns/characteristics available for better traffic control and management. This paper surveys the state-of-the-art in traffic engineering for SDNs, and mainly focuses on four thrusts including flow management, fault tolerance, topology update, and traffic analysis/characterization. In addition, some existing and representative traffic engineering tools from both industry and academia are explained. Moreover, open research issues for the realization of SDN traffic engineering solutions are discussed in detail.

1. Introduction

Traffic engineering (TE) is an important mechanism to optimize the performance of a data network by dynamically analyzing, predicting, and regulating the behavior of the transmitted data. It has been widely exploited in the past and current data networks, such as ATM and IP/MPLS networks. However, these past and current networking paradigms and their corresponding TE solutions are unfavorable for the next generation networking paradigms and their network management due to two main reasons. First, today's Internet applications require the underlying network architecture to react in real time and to be scalable for a large amount of traffic. The architecture should be able to classify a variety of traffic types from different applications, and to provide a suitable and specific service for each traffic type in a very short time period (i.e., order of *ms*). Secondly, facing the rapid growth in cloud computing and thus the demand of massive-scale data centers, a fitting network management should be able to improve resource utilization for better system performance. Thus, new networking architectures and more intelligent and efficient TE tools are urgently needed.

The recently emerged Software Defined Networking (SDN) [1,2] paradigm separates the network control plane from the data forwarding plane, and provides user applications with a centralized view of the distributed network states. It includes three layers and interactions between layers as shown in Fig. 1. The details of the SDN architecture overview are explained as follows: There may be more than one SDN controller if the network is large-scale or a

* Corresponding author. Tel.: +1 404 894 6616.
E-mail addresses: ian@ece.gatech.edu (I.F. Akyildiz), ahyoung.lee@ece.gatech.edu (A. Lee), pu.wang@wichita.edu (P. Wang), min.ch.luo@huawei.com (M. Luo), wu.chou@huawei.com (W. Chou).

http://dx.doi.org/10.1016/j.comnet.2014.06.002
1389-1286/

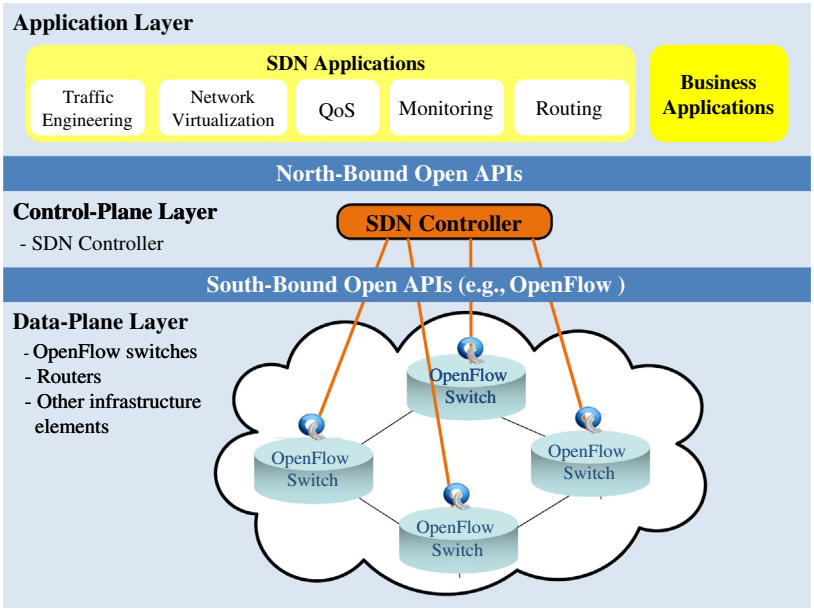


Fig. 1. Overview of SDN architecture.

wide-area region network. The control layer globally regulates the network states via network policies in either a centralized or distributed manner. Due to the unrestricted access to global network elements and resources, such network policies can be updated timely to react to the current flow activities. Furthermore, SDN applications exist in the application layer of the SDN architecture. A set of application programming interfaces (such as North-bound Open APIs) are supported to communicate between the application layer and the control layer in order to enable common network services, such as routing, traffic engineering, multicasting, security, access control, bandwidth management, quality of service (QoS), energy usage, and many other forms of the network management. In other words, these interfaces facilitate various business objectives in the network management. On the other hand, the data forwarding layer can employ programmable OpenFlow switches through OpenFlow controller, and the switches communicate with the controller via South-bound Open APIs (e.g., OpenFlow protocol) [1]. The OpenFlow (OF) protocol provides access to the forwarding plane of a network switch over the network and enables software programs running on OF switches to perform packet lookups and forwarding the packets among the network of switches or routers. These programmable switches follow the policies of the SDN/OF controller and forward packets accordingly in order to determine what path the packets will take through the network or switches or routers. In short, through the interactions among these layers, the SDN paradigm allows an unified and global view of complicated networks, and thus provides a powerful control platform for the network management over traffic flows. In the literature, most of the work so far is focused on developing the SDN architecture and with less effort on developing TE tools for SDN. While current TE mechanisms are extensively studied in ATM networks, IP-based and MPLS-based Internet, it is still unclear how these techniques perform under various traffic patterns, and how to obtain the enormous traffic and resource information efficiently in the entire network when the SDN is deployed. On the other hand, SDN promises to dramatically simplify the network management, reduce operating costs, and promote innovation and evolution in current and future networks. Such unique features of SDN provide great incentive for new TE techniques that exploit the global network view, status, and flow patterns/characteristics available for better traffic control and management. Therefore we first briefly discuss the classical TE mechanisms developed for ATM, IP and MPLS networks, and then survey in detail the state-of-the-art in TE for SDN from both academia and industry perspectives. Then, we examine some open issues in TE for SDN, and review some recent progresses in extending traditional TE techniques for SDN networks.

The remainder of the paper is organized as follows. Early TE issues and mechanisms based on ATM, IP and MPLS networks are given in Section 2. An overview of SDN traffic engineering solutions is provided in Section 3. From Section 4 to Section 7, the major SDN traffic engineering technologies, including flow management, fault tolerance, topology update, and traffic analysis, are presented, respectively. Existing TE tools for SDN with OF switches are further introduced in Section 8. The paper is concluded in Section 9.

2. Lessons learned from the past

Traffic engineering (TE) generally means that the network traffic is measured and analyzed in order to enhance

the performance of an operational network at both the traffic and resource levels [3].

In the late 1980s, ATM (Asynchronous Transfer Mode) networks were standardized in the telecommunications industry. At that time, the key objective of TE was to solve mainly the congestion control problem to meet the diverse service and performance requirements from multimedia traffic, due to the increasing demand for multimedia services (e.g., data, voice, and video).

At the end of the 1990s, IP-QoS routing technology became more influential over ATM switching, because the IP-QoS is much simpler and easier to configure in data networks. As a consequence, IP-QoS hit the market fast and dramatically increased the popularity of the services provided over the public Internet. In the late 90s, MPLS (Multiprotocol Label Switching) has emerged to work below IP and was an attempt to do simpler traffic engineering in the Internet, especially for the Internet backbones. However, TE for MPLS is still emphasized on the control and management of the Internet under current mechanisms and the network elements, because many control protocols, residing between application layer and link layer, are built on top of the Internet protocol suite, therefore failing to provide sufficient and efficient TE mechanisms for traffic control and management.

In this section, we review the TE concept and mechanisms from the historical perspective as shown in Fig. 2. Followed by that, we discuss the direction of TE for the new paradigm architecture of SDN networks.

2.1. ATM-based traffic engineering

In the late 1980s, Asynchronous Transfer Mode (ATM) has been developed and selected to enable the full use of the broadband integrated service digital networks (B-ISDN). ATM combines circuit switch routing of public telephone networks, packet switching of private data networks, and the asynchronous multiplexing of packets. ATM is a form of a cell switching using small fixed-sized packets and multiplexing technique that supports switching in public and private networks. ATM is capable of transporting multiple types of services simultaneously on the same network and all data are placed in cells of the uniform size. ATM communication is connection-oriented, which means that a connection must be established before any cells are sent.

In ATM networks, congestion control is critical to multimedia services (e.g., voice, data, and video) that are increasingly demanded and must meet the QoS requirements such as high- throughput, real-time, and low-latency. The congestion control schemes are categorized in two methods: reactive control and preventive control. Reactive control instructs the source nodes to throttle their traffic flow at the beginning of congestion by giving feedback to them. However, a major problem with reactive control in high-speed networks is slow with feedback because the reactive control is invoked for the congestion after it happens [4]. In preventive control schemes, unlike reactive control, the source nodes do not wait until congestion actually occurs. Instead, they try to prevent the network from reaching an unacceptable level of congestion. The most common and effective approach is to control traffic flow at entry points to the network (i.e., at the access nodes). This approach is especially effective in ATM networks because of its connection-oriented transport where a decision to admit a new traffic can be made based on the knowledge of the state of the route which the traffic would follow. Preventive control for ATM can be performed in three ways: admission control, bandwidth enforcement, and traffic classification.

2.1.1. Admission control

In the admission control, the network decides whether to accept or reject a new connection based on whether the required QoS requirement of the new request can be satisfied. When a new connection is requested, the network examines its service requirements (e.g., acceptable cell transmission delay and loss probability) and traffic

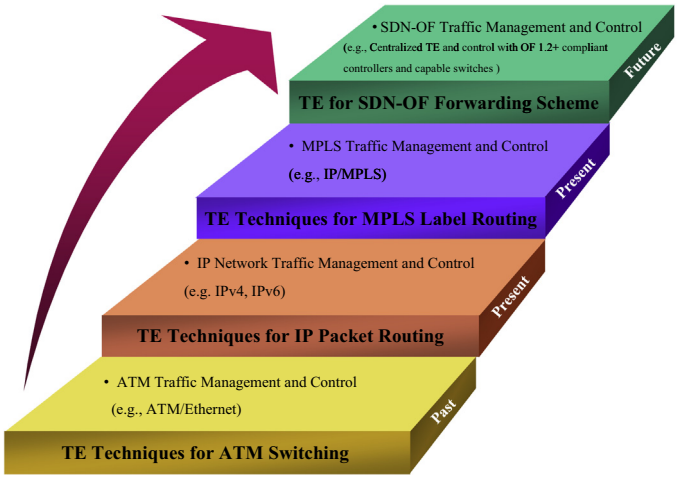


Fig. 2. Traffic engineering from past to future.

characteristics (e.g., peak rate, average rate, etc.). The network then examines the current load and decides whether or not to accept the new connection. The cell transmission delays and the cell loss probabilities are the most commonly applied decision criteria (QoS parameters) in the admission control. When the transmission delays and cell loss probabilities are applied in the admission control, their long-term-time-averaged values have been used [5]. Using a long-term-time-averaged value, however, may not be sufficient in an ATM network because the network traffic can change rapidly and dynamically, forcing the network to move from one degree of congestion to another. The effects of statistical traffic parameters are investigated on network performance in [6,5,7,8], such as the average burst length of the traffic sources, the peak rate of each source, and the number of sources.

2.1.2. Bandwidth enforcement

The traffic volume may be exceeded at the call setup, which easily overloads the network. In this case, the admission control alone is not sufficient to handle it, thus letting the exceeded traffic volume to become an "elephant cell". After a connection is accepted, the traffic flow of the connection must be monitored to ensure that the actual traffic flow conforms to the specified parameters during the call establishment. Therefore, the bandwidth enforcement mechanism is implemented at the edges of the network. Once an "elephant cell" is detected, the traffic flow is enforced by discarding and/or buffering the elephant cells.

The Leaky Bucket method [9] is one of the typical bandwidth enforcement mechanisms used for ATM networks to enforce the average bandwidth and the burst factor of a traffic source. One possible implementation of a Leaky Bucket method is to control the traffic flow by means of tokens, in which a queuing model is used. When an arriving cell enters a queue, it will be discarded if the queue is full. To enter the network, a cell must first obtain a token from the token-pool. If there is no token left, it must wait in the queue until a new token is generated. In the Leaky Bucket method, the elephant cells are either discarded or stored in a buffer even when the network load is light. Thus, the network resources are wasted. To avoid this problem, the marking method is proposed in [10]. In this scheme, elephant cells, rather than being discarded, are permitted to enter the network with violation tags in their cell headers. These elephant cells are discarded only when they arrive at a congested node. If there are no congested nodes along the routes, the elephant cells are transmitted without being discarded. Therefore, the total network throughput can be improved by using the marking method.

2.1.3. Traffic classification

ATM networks must support diverse service and performance requirements. Different traffic streams may have different delay requirements, even within delay-sensitive traffic (e.g., voice or video). To support multiple classes of traffic in ATM networks, priority mechanisms can be used, rather than uniform control mechanisms, which mean that different priority levels are given to different classes of traffic. There are two ways to use priorities: one can use a priority mechanism as a scheduling method (i.e., queuing discipline). In this way, different delay requirements can be satisfied by scheduling delay-sensitive or urgent traffic first. The second way is to use a priority scheme to control congestion. In this case, when a network congestion occurs, different cell loss requirements can be satisfied by selectively discarding (low priority) cells. Two dynamic priority schemes, Minimum Laxity Threshold (MLT) and Queue Length Threshold (QLT) [11], try to reduce the performance degradation for the low priority traffic. In these dynamic priority schemes, priority level changes with time. Also priority mechanism can be used as local congestion control schemes to satisfy different cell loss requirements of different classes of traffic. In [12], various traffic management and congestion control schemes have been proposed for ATM networks. It seems that there is no single preferred management method. In general, depending on the chosen scheme, there are tradeoffs between, the buffer resources and delay, buffer resources and overhead, or buffer resources and complexity or cost [13].

2.1.4. Learning from the ATM-based traffic engineering

From the brief review of traffic engineering on ATM networks and learning from the past, we believe that the SDN controller(s) must include a variety of congestion control and traffic management schemes, and admission control policy rules, to support different traffic types from different applications with different QoS requirements such as real-time applications (e.g., voice or video) or non-real-time applications (e.g., data), and have to consider the tradeoffs between load-balance and QoS in the network.

2.2. IP-based traffic engineering

Traffic engineering is an important feature for Internet providers trying to optimize the network performance and the traffic delivery. Routing optimization plays a key role in traffic engineering, i.e., finding efficient routes to achieve the desired network performance [14]. In [3], Internet traffic engineering is defined as large-scale network engineering which deals with IP network performance evaluation and optimization. Typically, the objectives of traffic engineering include balancing the load distribution and minimizing the bandwidth consumption in the network, which are similar to ATM-based traffic engineering as discussed above [14].

In IP networks, the quality of service (QoS) and resilience schemes are also considered as major components of traffic engineering. Because a variety of new multimedia applications not only have bandwidth requirements, but also require other QoS guarantees, such as end-to-end delay, jitter, packet loss probability, as well as energy efficiency. In addition, the fast resilience schemes are required to deal with different types of network failures (e.g., network node or link failure) that frequently may happen in IP networks [15]. In this case, the traffic engineering solutions must consider how to minimize the impact of failures on network performance and resource utilization. So far, the most of IP-based traffic engineering solutions in [16–18], have basic routing schemes that are based on

the shortest path and load-balancing schemes with the equally split traffic into equal cost multiple paths.

2.2.1. The shortest path routing

The basic idea of shortest path routing [19] is to set the link weights of interior gateway protocols (IGPs) according to the given network topology and traffic demand to control intra-domain traffic in order to meet the traffic engineering objectives. Most large IP networks run interior gateway protocols (IGPs) such as Open Shortest Path First (OSPF) or Intermediate System–Intermediate System (IS–IS) that select paths based on static link weights (such as cost value assigned at each link). Routers use these protocols to exchange link weights and construct a complete view of the topology inside the autonomous system (AS). Then each router computes shortest paths and creates a table that controls the forwarding of each IP packet to the next hop in its route [17]. However, shortest path routing does not seem flexible enough to support traffic engineering in a network supporting a different set of applications. In addition, the changes of static link weight may affect the routing patterns of the entire set of traffic flows (such as link failure). Selecting good link weights depends on having a timely and accurate view of the current state of the network. Thus, the Simple Network Management Protocol (SNMP) provides information about the status of the network elements, either by polling or via traps. In addition, it is possible to deploy IGP route monitors that track the topology and IGP parameters in the operational network. The operator also needs an estimate of the traffic volume between each pair of routers.

2.2.2. The equal-cost multi-path routing

In equal-cost multi-path routing [20], large networks are typically divided into multiple OSPF/IS–IS areas. In some cases, the network may have multiple shortest paths between the same pair of routers. The OSPF and IS–IS protocol specifications do not dictate how routers handle the presence of multiple shortest paths, because the IGP routing algorithm using static link does not have the flexibility to divide the traffic among the shortest paths in arbitrary proportions. Thus, routing based on link weights is not flexible enough to represent all possible solutions to the routing problem. Because of the dynamic traffic demands, the traffic volumes fluctuate over time in practice, and unexpected failures can result in changes to the network topology.

In addition, acquiring an exact estimate of the traffic matrix is difficult. The practical OSPF [21] provides shortest-path-first routing with simple load balancing by Equal-Cost Multi-Path (ECMP) that enables the traffic to split evenly amongst equal cost paths. More specifically, ECMP, based on the Hash function, aims to divide the hash space into equal-size partitions corresponding to the outbound paths, and forwards packets based on their endpoint information along the path whose boundaries enveloping the packets hash value. Although these schemes provide a good performance when operating with static load balancing, they are unsuitable for the dynamic load balancing protocols [22], since this static mapping of flows to paths does not account for either current network utilization or flow size, which results in collisions that overwhelm router or switch buffers so that the overall network utilization is degraded [23].

2.2.3. Learning from the IP-based traffic engineering

Today's IP data networks are far more complex and difficult to manage due to their data plane, control plane, and management plane are split and distributed across different network elements [24,25]. To encounter these problems, a 4D architecture as in Fig. 3 is introduced in [24], which completely separates the routing decision logic from the protocols that control the interaction between the network elements. The core components of the 4D architecture include the decision plane for a network-wide view of the network, the data plane for forwarding traffic, the discovery and dissemination planes for a direct control. In addition, the Routing Control Platform (RCP) [25] is introduced, RCP is a logically centralized platform that separates the IP forwarding plane to provide the scalability in order to avoid the complexity problems in the internal Border Gateway Protocol (iBGP) architectures. These ideas inspire the confidence of SDN researchers and system developers for a logically separated network with the SDN controllers and OF switches.

2.3. MPLS-based traffic engineering

Multi-Protocol Label Switching (MPLS) [26,27] was introduced as an attractive solution to traffic engineering by addressing the constraints of IP networks. MPLS-based TE can provide an efficient paradigm for traffic optimization. Most advantages of MPLS traffic engineering rely on the fact that it can efficiently support the explicit routing between source and destination, and thus can arbitrarily split traffic through the network, and highly flexible for both routing and forwarding optimization purposes [3]. In MPLS-based TE, the routers use the MPLS label-switching paradigm where labels are assigned and distributed between routers using the Label Distribution Protocol (LDP). Packets are assigned with labels by the ingress router, and then the packet is forwarded across the network

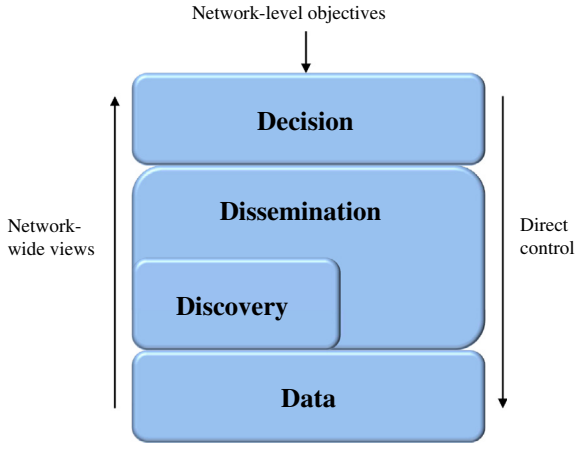


Fig. 3. 4D architecture [24].

using the label switching based on the label, rather than on the IP header information. At the egress router, the label is removed and the packet is again forwarded as an IP packet. After full label information is exchanged in the MPLS network, a Label Switching Path (LSP) is selected between all routers.

2.3.1. LSP tunnels

One significant feature of MPLS-based TE is the so-called LSP tunnels that are established by a signaling protocol such as the Resource Reservation Protocol (RSVP). When using the RSVP, the full QoS offerings of integrated services are made available, because the use of RSVP for differentiated services (DiffServ) is already defined within the Internet Engineering Task Force (IETF). The network resources can be allocated by multiple LSP tunnels that can be created between two nodes, and the traffic between the nodes is divided among the tunnels according to some local policy. However, the scalability and robustness become issues in MPLS-based TE [28], since the aggregate traffics are delivered through dedicated LSPs. The total number of LSPs within an intra-domain such as a "full mesh" network is $O\left(N^2\right)$ where N is the number of ingress and egress routers within a single domain [3], which is generally considered to be non-scalable with respect to network protocols [29]. In addition, the path protection mechanisms (e.g., using backup paths) are necessary in MPLS-based TE, as otherwise the traffic cannot be automatically delivered through alternative paths if any link failure occurs in active LSPs [3]. The network management is an important aspect of traffic engineering over MPLS. The success of the MPLS approach to traffic engineering eventually depends on the easiness with which the network can be observed and controlled.

2.3.2. Learning from the MPLS-based traffic engineering

The simplicity of the SDN can alleviate the complexities of the MPLS control plane with scalability and efficiency at the same time [30]. The implementation of OF with MPLS provides much easier and more efficient network management. Thus, the extension of OF switches with MPLS [31,30], simply match and process the MPLS flows, without requiring the MPLS per packet processing operations.

3. Overview of SDN traffic engineering

Traffic engineering mechanisms in SDN can be much more efficiently and intelligently implemented as a centralized TE system compared to the conventional approaches such as ATM-, IP-, and MPLS-based TEs because of the major advantages of the SDN architecture. More specifically, SDN provides (1) centralized visibility including global network information (e.g., network resource limitations or dynamically changing the network status) and global application information (e.g., QoS requirements); (2) the programmability without having to handle individual infrastructure elements, i.e., OF switches at the data plane can be proactively programmed and dynamically reprogrammed by the centralized controller to optimally allocate network resources for network congestion avoidance and enhanced QoS performance; (3) openness, where data plane elements (i.e., OF switches), regardless of the vendors, have a unified interface open to the controller for data plane programming and network status collection; and (4) multiple flow table pipelines in OF switches can make flow management more flexible and efficient.

Since the emergence of SDN, it has been applied to a variety of network environments, (i.e., Enterprise networks, large-scale data center networks, WiFi/cellular networks, etc.). TE technology is of critical importance to the evolution and success of SDNs. As shown in Fig. 4, current traffic engineering mechanisms mainly focus on four thrusts including flow management, fault tolerance, topology update, and traffic analysis including characterization.

First, according to the basic operation of flow management in SDNs, when a flow arriving at switch does not match any rules in the flow table, it will be processed as follows: (1) the first packet of the flow is sent by the ingress switch to the controller, (2) the forwarding path for the flow is computed by the controller, (3) the controller sends the appropriate forwarding entries to install in the flow tables at each switch along the planned path, and (4) all subsequent packets in the flow or even different flows with matching (or similar) attributes are forwarded in the data plane along the path and do not need any control plane action. In this operation, if the aggregated traffic consists of high number of new flows, a significant overhead can be yielded at both the control plane and data plane. Moreover, the forwarding rule setup can also take time, so that the latency can be increased. Therefore, to solve these problems, traffic engineering mechanisms for the flow management should be designed to address the tradeoffs between the latency and load-balance.

Second, to ensure the network reliability, SDN should have a capability to perform failure recovery transparently and gracefully, when a failure occurs in the network infrastructure (i.e., controllers, switches and links) [32]. Moreover, a single link or node failure should be recovered within 50 ms in carrier grade networks [32]. To increase the networking resiliency of SDN, in OF v1.1+, a fast failover mechanism is introduced for link or node failures, in which an alternative port and path can be specified, enabling the switch to change the forwarding path in the policy based routing without requiring a round trip to the controller. Although the situation is much improved with centralized network management, achieving fast failure recovery is still very challenging in SDN, because the central controller in restoration must calculate new routes and notify all the affected switches about the recovery actions immediately. Moreover, the failure recovery needs to consider the limited memory and flow table resources at switches.

Third, the topology update mechanism in SDNs focuses on the planned changes such as the network policy rule changes, instead of the network element or link failures. Since the centralized controllers manage all switches in SDN/OF networks by dynamically configuring the global network policy rules, a certain level of required consistency of the network policies needs to be guaranteed across the switches so that each individual packet or flow should be handled by either the old policy or the new

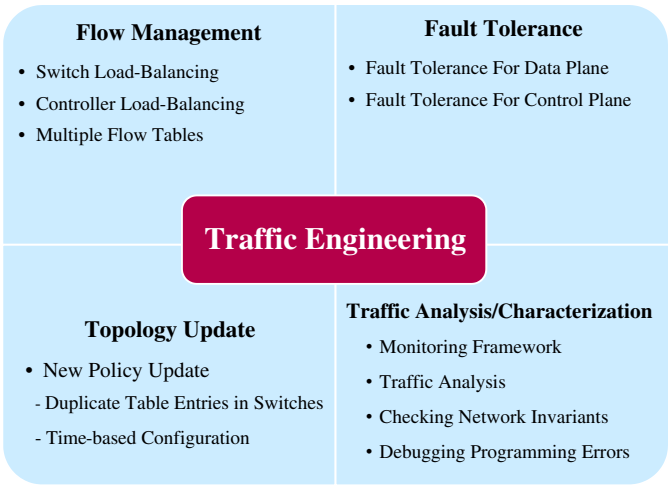


Fig. 4. The scope of traffic engineering approaches in current SDNs.

policy, but not by the conflicting combinations. Moreover, during the policy updating time, the affected flows may be dropped or delayed, which degrades the network QoS performance or leads to wasting network resources. Therefore, the key challenge in topology update is how SDN controller can efficiently update the network with required consistency in (near) real time. This would be even more challenging for a large SDN/OF network, where not every switch can be directly connected to the central controller.

Last but not least, the traffic analysis mechanisms should include traffic/network monitoring tools, network invariant checking mechanisms, programming error debugging software, flow/state data collection, analytics/mining of patterns/characteristics, etc. In particular, traffic/network monitoring tools are the most important prerequisite for traffic analysis and they are closely related to all other traffic analysis mechanisms, especially for detecting the network or link failures, and predicting link congestion or bottleneck. However, many SDN architectures use the existing flow based network monitoring tools from traditional IP networks. These methods can lead to high monitoring overhead and significant switch resource consumption [33]. Even though OF v1.3 introduced the flow metering mechanisms, most of the current controllers (e.g., NOX, POX, Floodlight, etc.) and available switches still do not provide an adequate support for different flow or aggregate statistics. In addition, the implementation of a controller with complex monitoring and analytical functionalities may significantly increase the design complexity [34]. Therefore, new traffic monitoring tools have to be developed to achieve low complexity, low overhead, and accurate traffic measurements.

4. Flow management

In SDN, when an OF switch receives a flow that does not match any rule in the flow entry at a switch, the first packet of the flow is forwarded to the controller. Accordingly, the controller decides whether it is required to install a new forwarding rule in the switches, which can lead to the balanced traffic load in the network. However, this forwarding rule installation process may take time and yield delay spikes. Moreover, if a high number of new flows are aggregated at the end of switches, significant overhead can be yielded at both the control plane and data plane. Thus, in this section we survey solutions that aim to avoid this bottleneck in SDN by considering the tradeoffs between latency and load-balance. The solutions are described in the following subsections including switch load-balancing, controller load-balancing, and multiple flow tables.

4.1. Switch load-balancing

4.1.1. Hash-based ECMP flow forwarding

The hash-based Equal-Cost Multi-Path (ECMP) [20] is a load-balancing scheme to split flows across available paths using flow hashing technique. The ECMP-enabled switches are configured with several possible forwarding paths for a given subnet. When a packet with multiple candidate paths arrives, it is forwarded on the one that corresponds to a hash of selected fields of that packet's headers modulo the number of paths [20,23], thus splitting load to each subnet across multiple paths [23].

A key limitation of ECMP is that two or more large, long-lived flows can collide on their hash and end up on the same output port, creating a bottleneck. This static mapping of flows to paths does not account for either current network utilization or flow size, thus resulting in collisions that overwhelm switch buffers and degrading overall switch and link utilization [23]. To encounter such problem, two load-balancing solutions, e.g., Hedera [23] and Mahout [33], are proposed. Table 1 presents the comparison of Hedera and Mahout schemes.

Hedera [23], is a scalable and dynamic flow scheduling system to avoid the limitations of ECMP. It has a global view of routing and traffic demands, collects flow information from switches, computes non-conflicting paths for flows, and instructs switches to re-route traffic accordingly. In the Hedera architecture, Hedera has a control loop of two basic steps. (1) When it detects large flows

Table 1
Qualitative overview of Hash-based ECMP flow forwarding schemes.

Hash-based ECMP flow forwarding schemes			
Proposed approaches	Elephant flow detection	Process overhead	Bandwidth overhead
Hedera [23]	Edge-switch	Between the controller and the switches	High at the switches
Mahout [33]	End-host	Between the switches and the hosts	High at the hosts

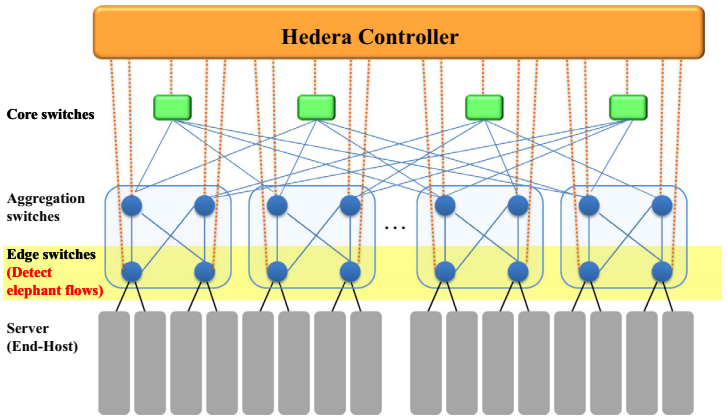


Fig. 5. Hedera control architecture in a fat-tree network. It detects large flows (such as elephant flows) at the edge switches for flow management.

("elephant" flows) at the edge switches (as depicted in Fig. 5) – e.g., a new flow event occurs, the switch forwards it along one of its equal-cost paths, based on a hash on the flow's 10-tuple. This path is used until the flow grows and meets a specified threshold rate (such as 100 Mbps in Hedera implementation). (2) It estimates the natural demand of large flows and computes good paths for them. If the flow grows past the threshold rate, Hedera dynamically calculates an appropriate path for it and installs these paths on the switches. Hedera uses periodic polling scheduler at the edge switches for collecting flow statistics and detecting large flows every five seconds to achieve a balance between improving aggregate network utilization with minimal scheduler overhead on active flows.

Mahout [33] manages flow traffics by requiring timely detection of significant flows ("elephant" flows) that carry large amount of data. The existing elephant flow detection methods, such as periodic polling of traffic statistics (e.g., NetFlow [35]) from switches or sampling packets (e.g., sFlow [36]) from switches, have high monitoring overheads, incurring significant switch resource consumption, and/or long detection times [33]. Hedera uses periodic polling for elephant flow detection that pulls the per-flow statistics from each of its edge-switch. However, the edge-switch may need to maintain and monitor over 38,400 flow entries if with 32 servers, each server generates 20 new flows per second with a default flow timeout period of 60 s, and it becomes infeasible in the real switch implementations of OF. To address these problems, the key idea of Mahout is that it monitors and detects elephant flows at the end host via a shim layer in the Operating System (as depicted in Fig. 6), instead of directly monitoring the switches in the network. In Mahout, when the shim layer detects that the socket buffer of the flow crosses a chosen threshold, the shim layer determines that the flow is an elephant. Then, it marks subsequent packets of that flow using an in-band signaling mechanism. The switches in the network are configured to forward these marked packets to the Mahout controller. At this time, the Mahout controller computes the best path for this elephant flow and installs a flow-specific entry in the rack switch, otherwise the switches perform the ECMP forwarding action as a default. This simple approach allows the controller to detect elephant flows without any switch CPU- and bandwidth-intensive monitoring. This simple approach ensures that the flows are bottlenecked at the application layer and not in the network layer. MicroTE [37] is a very similar approach as Mahout. It is a traffic engineering scheme to detect significant flows at the end hosts so that when a large portion of traffic is predicted, then MicroTE routes them optimally. Otherwise, the flows are managed by the ECMP scheme with heuristic threshold.

4.1.2. Wildcard rule flow forwarding

OF uses a field of 32-bit (in v1.0-1.1) or 64-bit (in v1.2-1.3) wildcards that have binary flags in the match. Thus, using OF flow-match wildcards can reduce the control-plane load [38]. OF is a great concept that simplifies the network and traffic management in enterprise and data center environments by enabling flow-level control over Ethernet switches and providing global network visibility [39]. However, the central control and global visibility over all flows require the controller to setup all flows for the critical path in the network, which is not sufficiently scalable, because using a single central controller for all flow setups causes both network load bottleneck and latency [40]. To encounter such problem, the following two

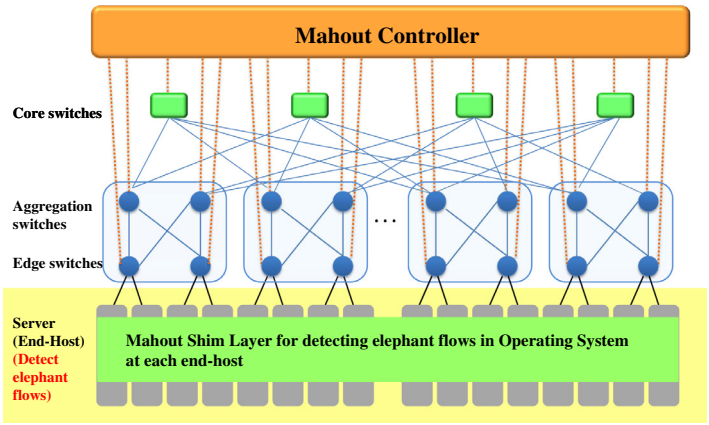


Fig. 6. Mahout control architecture in a fat-tree network. It detects large flows (such as elephant flows) at the end of host servers for flow management.

solutions, DevoFlow and DIFANE, have been proposed [40,39,41].

DevoFlow [40,39] is proposed for reducing the number of interactions between the controller and switches. This mechanism implements wildcard OF rules at switches, so that the switches can make local routing decisions with matching microflows and the controller maintains the control over only targeted "significant flows" (such as "elephant" flows) that may be QoS-significant flows. Similarly, an efficient load-balancing architecture is proposed in [42] which employs a partitioning algorithm for proactively generating wildcard rules, which are installed in the switches to handle the requests for "microflows" without involving the controller. Then, the switch performs an "action" of rewriting the server IP address and forwarding the packet to the output port.

DIFANE [41] proposed a distributed flow architecture for enterprise networks using wildcard rules in the switches, in such a way that only the switches handle all data packets in the data plane. For example, if the arrival traffic flows do not match the cached rules in the ingress switch, then the ingress switch encapsulates and redirects the packet to the appropriate authority switch based on the partition information. The authority switch handles the packet in the data plane and sends feedback to the ingress switch to cache the relevant rules locally. Also, for minimizing overhead at the controller, DIFANE uses the link-state routing that enables the switches to learn about the topology changes without involving the controller, and adapts routing quickly. However, this approach may have a heavy load on the core switches and they do not provide a load-balancing scheme in their architecture.

4.2. Controller load-balancing

Whenever a flow is initiated in the network, the OF switch must forward the first packet of the flow to the controller for deciding an appropriate forwarding path. Such a unique feature of SDN makes the centralized controller become another performance bottleneck, in addition to heavy traffic load among switches mentioned in Section 4.1. In particular, a single and centralized controller cannot work efficiently, as the whole network grows because of the increased number of network elements or traffic flows. Furthermore, while only providing one type of service guarantees, this single controller fails to handle all different incoming requests. For example, as shown in [43], a current NOX control platform can only handle 30 K flow initiations per second with around 10 ms for each flow install time. This serving ability is insufficient for SDN applications, especially for the data center scenarios. Therefore, by designing different deployments of possible multiple controllers, several promising solutions are proposed to avoid this bottleneck between the controllers and OF switches, and their results are summarized in Fig. 7. In the following, we classify these controller deployment solutions into four categories: (1) logically distributed controller deployment, (2) physically distributed controller deployment, (3) hierarchical controller deployment, and (4) hybrid controller deployment. Table 2 presents the comparison of different schemes of controller load-balancing. Other solutions are described in the following subsections including the multi-thread controllers and the generalized controllers for the controller load-balancing.

4.2.1. Logically distributed controller deployment

HyperFlow [44] is a distributed event-based control plane for the OF network as a SDN paradigm, which use OF protocol to configure the switches. Specifically, the Hyper-Flow can realize a logically centralized network control by using physically distributed control plane in order to address the scalability while keeping the benefit of network control centralization. HyperFlow localizes the decision making to individual controllers for minimizing the control plane response time to data plane requests, and provides scalability while keeping the network control logically centralized. Through the synchronization schemes, all the controllers share the same consistent network-wide view and locally serve requests without actively contacting any remote node, thus minimizing the flow setup times. More specifically, the HyperFlow-based network is composed of OF switches as forwarding elements, NOX controllers as decision elements, each of which runs an instance of the HyperFlow controller application, and an event

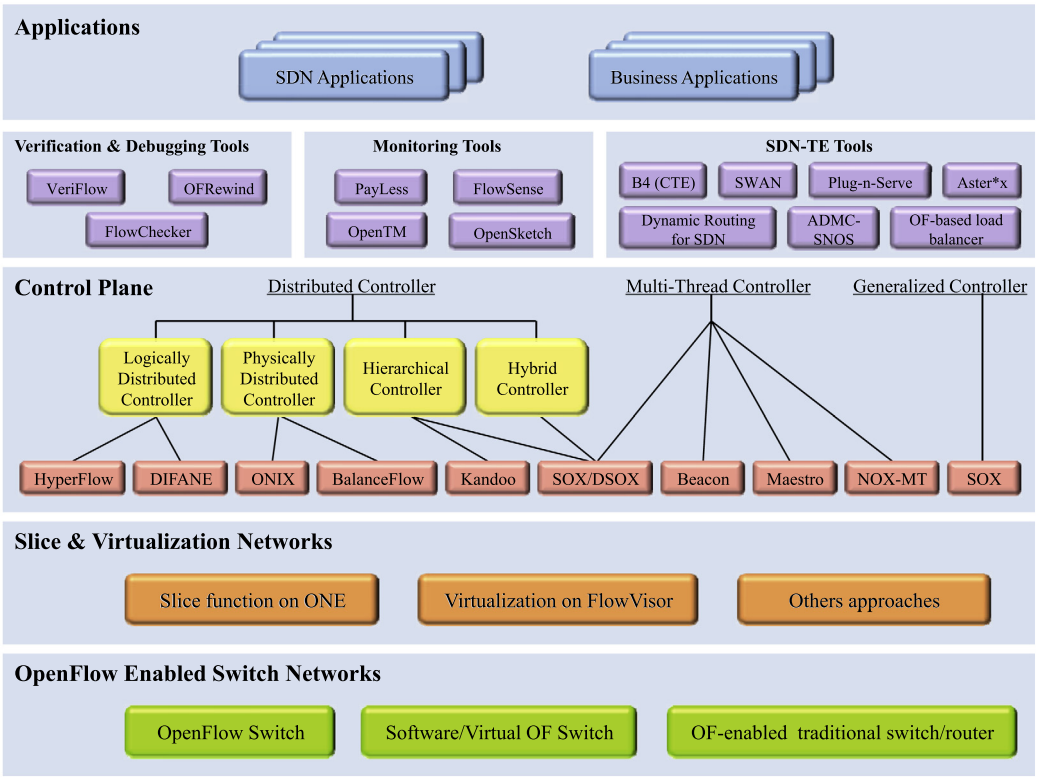


Fig. 7. SDN stack.

propagation system for cross-controller communication. Each switch is connected to the best controller in its proximity. All the controllers have a consistent network-wide view and run as if they are controlling the whole network. Towards this, the HyperFlow uses publish/subscribe system to provide persistent storage of published events using WheelFS [51] for minimizing the cross-site traffics required to propagate the events (i.e., controllers in a site should get most of the updates of other sites from nearby controllers to avoid congesting the cross-region links).

DIFANE [41] has the following two main ideas: (1) The controller distributes the rules across a subset of the switches, called authority switches, to scale to large topologies with many rules. (2) The switches handle all packets in the data plane, i.e., TCAM (the Ternary Content Addressable Memory), at a switch and divert packets through authority switches as needed to access the appropriate rules. The rules for diverting packets are naturally expressed as TCAM entries. The DIFANE architecture consists of a controller that generates the rules and allocates them to the authority switches. Authority switches can be a subset of existing switches in the network, or dedicated switches that have larger memory and processing capability. Upon receiving traffic that does not match the cached rules, the ingress switch encapsulates and redirects the packet to the appropriate authority switch based on the partition information. The authority switch handles the packet in the data plane and sends feedback to the ingress switch to cache the relevant rule(s) locally. Subsequent packets matching the cached rules can be encapsulated and forwarded directly to the egress switch. Using link-state routing to compute the path to the authority switch, all data plane functions required in DIFANE can be expressed with three sets of wildcard rules. (1) Cache rules are the ingress switches cache rules so that most of the data traffic hits in the cache and is processed by the ingress switch. The cache rules are installed by the authority switches in the network. (2) Authority rules are only stored in authority switches. The controller installs and updates the authority rules for all the authority switches. When a packet matches an authority rule, it triggers a control-plane function to install rules in the ingress switch. (3) Partition rules are installed by the controller in each switch. The partition rules are a set of coarse-grained rules. With these partition rules, a packet will always match at least one rule in the switch and thus always stay in the data plane. Since all functionalities in DIFANE are expressed with wildcard rules, DIFANE does not require any data-plane modifications to the switches and only needs minor software extensions in the control plane of the authority switches. Thus, DIFANE is a distributed flow management architecture that distributes rules to authority switches and handles all data traffic in the fast path.

4.2.2. Physically distributed controller deployment

Onix [45] is a distributed control platform which runs on a cluster of one or more physical servers. As the control platform, Onix is responsible for giving the control logic

Table 2
Qualitative overview of difference schemes of controller load-balancing for different types of distributed controllers.

Controller load-balancing				
Type of Distribution controllers	Description	Proposed approaches	Summery	Disadvantage
Logically distributed controller deployment	A logically centralized and physically distributed control plane.	HyperFlow [44]	• Publish-subscribe method with WheelFS file system for cross-controller communication and global network view sharing.	• Additional maintenance and subscription management overhead.
		DIFANE [41]	• Distributed controller's rules across a subset of the authority switches.	• Small overhead between the central controller and switches and high resource consumption (i.e., CPU, TCAM space) at switches.
Physically distributed controller deployment	Control platforms distributed on one or more servers.	Onix [45]	• Publish-subscribe method with the NIB database system.	• Additional maintenance and subscription management overhead.
		BalanceFlow [46]	• One super controller and many normal controllers, where the super controller is responsible for load balancing among all controllers.	• Additional overhead at control plane.
Hierarchical controller deployment	Two-level hierarchy for controllers (local controllers and a logically centralized root controller).	Kandoo [47]	• Local controllers execute local applications and each local controller controls one or some switches. • The root controller controls all local controllers and runs non-local control applications.	• No global network view for the application processes at local controllers.
Hybrid controller	Logically Centralized, but physically distributed clusters of controllers.	SOX/DSOX [48–50,85]	• Centrally controlled cluster of controllers running in equal mode with automatic failover and load balancing while such a controller cluster is targeted to manage a "significant-size" of a (sub) network. • The controller clusters can be physically distributed to control different (sub) networks with required synchronization for necessary consistency, while those distributed controllers can be inter-connected through a service bus or extended BGP protocol as defined in the software-services defined networking technology.	• No full consistency among the distributed controller clusters.

programmatic access to the network (such as read and write forwarding table entries). The network control logic is implemented on top of Onix's API from their previous version [52], which determines the desired network behavior. So the core part of Onix is a useful and general API for network control that allows for the development of scalable applications. Using the Onix's API, a view of the physical network, control applications can read and write state to any element in the network, hence keeping state consistent between the in-network elements and the control application that runs on multiple Onix servers. The copy of the network state tracked by Onix is stored in a data structure named the Network Information Base (NIB), which is a graph of all network entities within a network topology. Therefore, Onix provides scalability and reliability by replicating and distributing the NIB data between multiple running controller instances.

BalanceFlow [46], which employs a similar concept of distributed controllers from Onix, is a controller load balancing architecture for wide-area OF networks, which can partition control traffic load among different controller instances in a more flexible way. BalanceFlow focuses on controller load balancing that (1) flow-requests will be dynamically distributed among controllers to achieve quick response, and (2) the load on an overloaded controller will be automatically transferred to appropriate low-loaded controllers to maximize the controller utilization. They presented Controller *X* action for an alternative flow-requests spread mechanism, which can be implemented in OF switches. Using a more flexible way, the controllers can reactively or proactively install fine-grained or aggregated flow entries with Controller *X* action on each switch. Different flow-requests of each switch can be allocated to different controllers. All controllers in BalanceFlow maintain their own flow-requests information and publish this information periodically through a cross-controller communication system to support load balancing. There are two types of controllers in BalanceFlow network, one super controller and many normal controllers. The super controller is responsible for balancing the load of all controllers, it detects controller load imbalance when the average number of flow-requests handled by a controller is larger than some threshold of the total flow-requests rate in the network. The threshold is adjustable according to the performance of the super controller, the number of controllers, and the network environment.

4.2.3. Hierarchical controller deployment

Kandoo [47] creates a two-level hierarchy for controllers: (1) local controllers execute local applications as close as possible to switches (i.e., applications that process events locally), and (2) a logically centralized root controller runs non-local control applications (i.e., applications that require access to the network-wide state). A network controlled by Kandoo has multiple local controllers and a logically centralized root controller. These controllers collectively form Kandoo's distributed control plane. Each switch is controlled by only one Kandoo controller, and each Kandoo controller can control multiple switches. If the root controller needs to install flow-entries on switches of a local controller, it delegates the requests to the respective local controller.

4.2.4. Multi-thread controllers

To enhance the request processing throughput, multi-threaded multi-core SDN controllers have been developed, which exploit parallelism (i.e., multi-core) architecture of servers to provide high throughput with scalability at controller. Table 3 gives a qualitative overview of several multi-thread controllers, and the results of each proposed approach depends on their testbed conditions. The detailed description of each controller is given as follow.

Maestro [53] is a multi-threaded SDN controller implemented in Java. Maestro control platform is based on a server machine with the total 8 cores from two Quad-Core AMD Opteron-2393 processors with 16 GB of memory. Actually 7 cores are used for worker threads and one processor core is used for functionalities (such as Java programming class management and garbage collection). In the performance evaluation, the throughput measures a response time for flow request message sent by its switch to the controller and it returns to its origin switch. Maestro achieves a maximum throughput of around 630,000 rps (responses per second) with an average delay around 76 ms.

Table 3
Quantitative overview of multi-thread controllers.

Multi-thread controllers				
Proposed approaches	OpenFlow version	Number of threads used in CPU cores	Maximum throughput	Average delay
Maestro [53]	v1.0.0	7 (8 cores from 2 × Quad-Core AMD Opteron 2393 processors)	0.63 million rps	76 ms
Beacon [54]	v1.0.0	12 (16 cores from 2 × Intel Xeon E5-2670 processors)	12.8 million rps	0.02 ms
NOX-MT [55]	v1.0.0	8 (8 cores from 2 GHz processor)	1.6 million rps	2 ms
SOX [48]	v1.3+	4 (4 cores from 2.4 GHz processor)	0.9 million pps per server; 3.4+ million pps with 4 servers in the cluster while hitting the I/O limit	N/A

For the completeness of the paper, we include performance numbers publically reported by vendors. It should be cautioned that, as a well known fact, all these numbers were reported with some specific tests designed by vendors, and no common tests, parameters, and environments are used so far. In addition, some controllers are very basic in functionality, and therefore would naturally demonstrate better performance.

Beacon [54] is a multi-threaded SDN controller implemented in Java, to provide a high performance with linear performance scaling. Beacon control platform is based on a server machine with total 16 cores from two Intel Xeon E5–2670 processors with 60.5 GB of memory. The (IBeaconProvider) interface is used to interact with the Beacon controller and OF switches. Beacon also provides additional Application Programming Interfaces (APIs) that are built on the core, which consists of a device manger interface (IDeviceManager) to search for devices (i.e., add, remove, or update devices); a topology interface (ITopology) to enable the retrieval of a list of links; an event registration to notify when links are added or removed; a routing interface (IRoutingEngine) that allows interchangeable routing engine implementations; a Web interface (IWebManageable) that allows the developers to implement an interface to add their own UI elements. In the performance evaluation where all the controllers with a single-thread in their testbed based on Cbench (controller benchmarker), it is shown that Beacon has the highest throughput at 1.35 millions rps, and is followed by NOX with 828,000 rps, and Maestro with 420,000 rps (even though Maestro is a multi-threaded controller, however, Maestro with a single-thread model has less throughput performance compared to NOX which is a single-thread model controller). In the second test where the controllers are configured with a different number of threads, Beacon running from 2 to 12 threads, has the maximum throughput 12.8 million rps, NOX with two to eight threads, can handle 5.3 million rps, and Maestro with its maximum of 8 threads achieves 3.5 million rps. The latency test shows that Beacon has the lowest average response time around 0.02 ms, while Maestro and NOX are similar performance each between 0.04 ms and 0.06 ms.

NOX-MT [55] is a multi-threaded SDN controller implemented in C++ based on NOX, which improves a single-threaded NOX's throughput and response time. NOX-MT uses well known optimization techniques including I/O batching for minimizing the overhead of I/O, and Boost Asynchronous I/O (ASIO) library for simplifying multi-threaded operation. In the experiment, NOX-MT control platform is based on an 8 core server machine, which handles about 1.8 million rps with an average response time around 2 ms. Beacon and Maestro have a similar maximum throughput achieved about 0.5 and 0.4 million rps respectively. In a single-thread in their testbed, the evaluation result shows that NOX-MT outperforms the others for maximum throughput achieved about 0.4 million rps, and followed by Beacon about 0.1 million rps, and both Maestro and NOX are almost same performance with lowest throughput about below 0.1 million rps.

4.2.5. Generalized controllers [48]

Since its first publication in 2009, OF protocols [1] have been evolving rapidly with the advances in SDN technologies. To enhance the flexibility, reliability, and advanced networking capabilities, the subsequent standard releases after OF v1.0, i.e. OF v1.1, v1.2, and v1.3+, gradually introduced many core functionalities such as multi-flow tables and multi-controllers, in addition to other critical needed features such as IPv6, MPLS, and flow metering. However, these desired new capabilities came with a cost, in terms of their renewed complexity and difficulty for efficient system architecture and implementation, for both the controllers and switches. Changes in the latest OF protocols are so significant and become incompatible with each other. It is not only because of those newly added features, but also the message meanings, formats, and parameters are also revised and modified. Moreover, in a foreseeable future, SDN/OF based technology will coexist and probably interoperate with existing IP based ones. The reality, however, is that to design an efficient and powerful controller for the later versions of SDN/OF protocols has not been that easy, and controllers currently from both the market place and open-source community typically support one or certain versions of the OF protocol. This would cause problems for adaptors of the SDN/OF technology as it can lead to repetitive investments and also possible isolated small networks fragmented by the non-compatible standard versions, causing substantially increased complexity and management difficulties. Therefore it would be advantageous to design an architecture that effectively and efficiently supports the internetworking of those different protocols and standards, with one core set of integrated components.

SOX [48], the Smart OF Controller (SOX) is a generalized SDN controller, developed and introduced in October 2012, to control SDN/OF based data networking with both OF v1.0 and v1.2 switches. Apart from being a generalized SDN/OF controller, SOX designers, for the first time in a large networking application, adopted and promoted the use of best software-engineering practice of model driven architecture (MDA) [48]. Introducing and applying MDA in SDN is aimed at improving the extensibility, modularity, usability, consistency, and manageability of SDN. The extensibility of SOX is demonstrated in its later extensions and enhancements to support networking with OF v1.0, v1.2, and v1.3 switches, as well as many new networking features such as interworking with routers with MPLS, MPLS/TE, differentiated QoS, and interworking through BGP with other networking domains [1].

In addition, SOX is multi-threaded and can be deployed on a clustered environment in equal-equal mode [48,56,57]. The number of threads/processes or controller instances in SOX are dynamically adjusted and fluctuated with the level of network traffic (packet-in rates) to the controller. New instances of controller will be added into the pool when the average load on existing controller instances climbs above a pre-set utilization threshold, and live controller instances will be decreased when the average load on the controller instances drops below a utilization level. This design offers a balance between the controller response time/scalability and the computing resource utilization. The average throughput for a single SOX server is 0.9 million pps (packet-in per second), while 4 servers in a cluster could reach 3.4+ million pps while hitting the I/O bottleneck, and further addition of servers to the cluster would not help. Each packet-in would generate N+1 responses, while N is the number of switches used for the given flow. Since they deploy 3 switches in a triangular topology, compared with a single switch for most of the other reported testing (vendors and universities), SOX's effective rps would be $4\times$ of the packet-in rate: single server 0.9 M $\times$ 4 = 3.6 million rps, 4 servers 3.4 + M $\times$ 4 = 13.2 million rps.

Table 4
Overview of hardware memories for implementing flow tables on OpenFlow switch.

	DRAM	SRAM	TCAM
Purpose	Store for all data-plane functions	Store for data-plane functions	Search for data-plane logic
Cost	Low (Costs per Mega-byte)	High (Costs per Mega-byte)	Very High (Costs per Mega-bits)
Scalability (Storage)	Very Large (Millions of flows)	Large (Hundreds of thousand of flows)	Small (A couple of thousands of flows)
Speed (for packet matching and distribution)	Slow	High	Very high
Throughput (Depending on speed)	Slow (A dozen of GbE ports or a couple of 10GbE ports at line rate)	High (A couple of dozens of GbE ports at line rate)	Very High (48GbE + 4 × 10GbE ports or 48 × 10GbE + 4X40GbE ports at line rate)

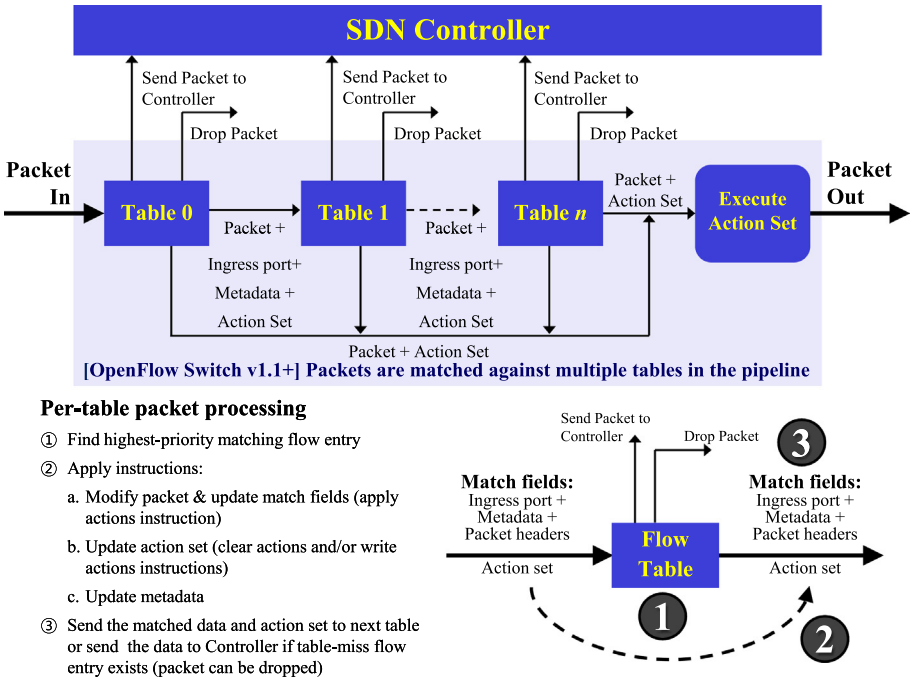


Fig. 8. Packet flow over multiple flow table pipelines.

4.3. Multiple flow tables

Other consideration is the multiple flow tables for flow management. Flows are defined by a sequence of packets for each flow from its origin to destination that share some common characteristics. At the beginning, the OF specification v1.0-based switches [58] have a single match table model typically built on TCAM. In this OF concept, a flow is identified by using its packet header field matching with a combination of at least 10-tuple including ingress port, VLAN id, Ethernet, IP, and TCP header fields. These aggregate fields are put into a single flow table in the TCAM of an OF switch. However, the single table for implementing flow rules creates a huge ruleset and could result in limited scale and inability for large scale deployments since TCAM space is limited and expensive resource as shown in Table 4. Also it is inefficient to store so many attributes in a single table with tremendous redundancy and slow in searching and matching.

To make flow management more flexible and efficient, OF v1.1[1] introduced the mechanism of multiple flow tables and actions associated with each flow entry, as shown in Fig. 8. OF switch can have one or more flow tables, when a packet arrives at the switch, first the switch identifies the highest priority matching flow entry, and then applies instructions or actions based on the flow fields. The action performs to send the matched data and action set to next appropriate table in the switch for pipeline processing. Unknown flows that do not match any flow entries in the multiple flow tables may be forwarded to the Controller or dropped. Thus, by decomposing the single flow table (with length of a flow entry with 40 or so attributes, exceeding 1000 bits with the OF v1.3+) into multiple more normalized set of tables, such a mechanism significantly improves TCAM utilization and also speeds up the matching process.

Additionally, OF v1.3+ supports Meter table for operating QoS requirements at each level of flows from user's

Table 5
Qualitative overview of hardware switch features with OpenFlow v1.3-enabled.

Hardware switch features								
Vendors	Product	Network interface	Max-switching capacity	Max-packet processing	Latency	OpenFlow v1.3-enabled (Dated by April 17 2014)		
						Support	Max-OpenFlow entries	Number of flow tables
Huawei [59]	SN-640 Switch Series	48 × 10GE + 4 × 40GE	1.28 Tbps	960 Mpps	300 ms ∼ 400 ms	OpenFlow v1.2/ v1.3 (Q2 2013)	630 K	More than 3 (9 stage pipelines)
HP [60]	HP 5900 Switch Series	48 × 1GE/ 10GE + 4 × 40GE, etc.	1.28 Tbps	952 Mpps	1.5 μs	OpenFlow v1.0/v1.3 (Q2 2013)	N/A	N/A
NEC [61]	PF5200 Switch	48 × 0.01GE/0.1GE/ 1GE + 4 × 1GE-OF/ 10GE	0.176 Tbps	131 Mpps	N/A	OpenFlow v1.0/ v1.3.1 (Q3 2013)	160 k	N/A
IBM [62]	G8264 Switch	48 × 1GE/ 10GE + 4 × 40GE	1.28 Tbps	960 Mpps	880 ms	OpenFlow v1.3.1 (with IBM Networking OS 7.8) (Q4 2014)	N/A	N/A
Pica8 [63]	PicOS-based Switch Series	1GE/10GE/40 GE	1.28 Tbps	960 Mpps	1.0 μs	OpenFlow v1.3 (Q1/ Q2 2014)	N/A	N/A
Broadcom [64]	StrataXGS Switch Series	32X40GE/(100+) X1GE/10GE, etc.	1.28 Tbps	N/A	N/A	OpenFlow v1.3.1 (Q1 2014/Ready to support)	N/A	8 (6 stage pipelines)
Brocade [65]	MLXe/ CER/CES Switch Series	10GE/40GE/100GE	25.6 Tbps	19 Bpps	N/A	OpenFlow v1.3 (Ready to support Jun 2014)	128 K	N/A

Another important comparison is whether the switches truly support multi-flow table pipelines optimized for application's scenarios. However, some switches have only one or few table(s).

demanded or different application's traffic. Some switch vendors have been started to develop OF v1.3-enabled switches as for traffic engineer to effectively handling flows while increasing performance, scalability, and flexibility in SDN paradigm. Table 5 presents a qualitative overview of hardware switch features with OF v1.3-enabled.

4.4. Open research issues

So far we discussed many flow management mechanisms developed for SDN networks. The majority of the proposed solutions focuses on load-balancing problem in both data and control planes. There are still many open research problems within the flow management in SDNs:

- *Dynamic load-balancing scheme for the data-plane layer:* In order to achieve load-balancing with low-latency network performance and avoiding network bottleneck in SDNs, we introduced two major flow forwarding approaches in Sections 4.1.1 and 4.1.2 based on Hash-based ECMP and Wildcard rule. The common objectives of both approaches are how to efficiently detect "elephant" flows, i.e., extremely large flows, by using the conventional ECMP scheme and the Wildcard rule. These can be implemented by the OF specification [1]. However, these load-balancing schemes are either static by setting heuristic fixed threshold at edge devices (e.g., Hash-based ECMP flow forwarding scheme) or of little adaptability to flow dynamics (e.g., Wildcard flow matching rule scheme). The effectiveness of load-balancing solutions are directly related to traffic characteristics and link capacities in a given network. For example, data center traffic can traverse through the edge, aggregation, and core links with different link capacities. It is identified that the data center traffic on edge and aggregation links is more bursty than that on core links. Such difference in traffic burstiness leads to high packet loss rates of the underutilized edge and aggregation links, and low packet loss rates of highly utilized core links [66,67]. Therefore, the traffic engineering in SDN demands a dynamic load-balancing mechanism that is dynamically adaptive to time-varying network states and fine-grained traffic characteristics such as traffic burstiness and inter-arrival times.
- *Dynamic load-balancing scheme for the control-plane layer:* There are two major deployments introduced in Sections 4.2.1, 4.2.2, 4.2.2 and 4.2.4 for distributed controllers to avoid a significant bottleneck at the single centralized controller in the large-scale SDN network. One is the hardware system-based mechanism that the controllers are distributed at different locations such as the physically separated servers, or controller's operations are split down the different-level hierarchy, and also including the hybrid controller approach. The other one is the operating system-based mechanism such as the multi-thread controllers. However, the load balancing schemes for control plane are largely unexploited. In particular, the control plane load balancing solutions

need to solve a set of fundamental problems, which aim to find the optimal number, locations, workload distribution, control message forwarding paths of SDN controllers in such a way that the optimal balance between the control message delay performance and control overhead/cost can be achieved subject to control/data message traffic statistics and network topology diversity. There exist very few papers that address the controller load balancing problem in the literature. In [68], the controller placement problem is investigated, where the distance between a controller and switches is adopted as the performance metric and several well-known network topologies are evaluated through simulations to find the optimal controller location. In [46], the controller placement problem is further investigated by proposing two heuristic algorithms to determine the proper number and locations of controllers with an objective to minimize the flow setup time and communication overhead. The controller workload distribution problem is studied in [46], where a heuristic algorithm is proposed to adjust the workload of each controller dynamically according to average flow-requests in all switches and switch-to-controller latency. Nevertheless, these efforts only look for quantitative or even heuristic results than qualitative analysis. In addition, there is lack of a thorough study to bring traffic statistics into control message load balancing.

- *Adaptive multi-flow table schemes:* The number of flows managed by a switch is limited by the size of its flow tables, because the scalability of using multiple flow tables is limited due to their very small size and high cost of TCAM space as shown in Table 4. In general, TCAM-based tables are limited to a few thousand entries. However, practically a single switch in data center can handle more than 100 million packet flows per second. Thus, the flexible and adaptive flow table methods should be developed so that the new flows exceeded from the limited space of TCAMs will be replaced to the large and lower cost SRAM or DRAM spaces. These methods should be associated with a traffic scheduling method for different QoS flows. Although some methods, such as the RMT (Reconfigurable Match Tables) model [69] and the FlowAdapter [70], have been proposed to address some challenging issues caused by the resource constraint of TCAM-based tables, there are some open issues with the implementation of a multi-flow table pipeline in current switching hardware. For example, how can multiple flow tables be converted efficiently to the different hardware capabilities? How can the optimal number of multiple flow tables in the pipeline be determined since it is dependent on switching hardware and also application scenarios?

5. Fault tolerance

To ensure network reliability, SDN should have a capability to perform failure recovery transparently and gracefully, when failures occur in the network infrastructure (i.e., controllers, switches and links) [32]. More specifically, as required by carrier grade networks, TE mechanisms must provide fast failure recovery so that carrier grade networks can detect and recover from incidents without significantly impacting users. In addition, even though a switch could identify the failed link, it has neither the intelligence nor the global knowledge to establish a new route. It must depend on the updates from the controller to establish an alternate route. Until a controller identifies a failed link and updates flow table entries in all the relevant switches, the packet that are supposed to travel on the failed link will be dropped. In the case of switch failure, even though the controller can sense the failure of a switch and the use of the fast failover mode could help switching the traffic to the protection path, but when the failed node comes back to work, it will still be the responsibilities of the controller to re-establish the network topology and the optimal routes for the on-going traffic.

Despite its great importance, achieving fast failure recovery, e.g., within 50 ms, is a quite challenging task for SDN, because the central controller in restoration must compute new routes and notify all the affected switches about a recovery action immediately. In this section, we investigate current research efforts on realizing fast failure recovery in SDN networks.

5.1. Fault tolerance for data plane

5.1.1. Data plane failure recovery mechanisms

There are two types of failure recovery mechanisms for the network element and link failures: restoration and protection [71,75,76].

- **Restoration**: the recovery paths can be either pre-planned or dynamically allocated, but resources are not reserved until failure occurs. When a failure occurs additional signaling is needed to establish the restoration path.
- **Protection**: the paths are pre-planned and reserved before a failure occurs. When a failure occurs, no additional signaling is needed to establish the protection path.

Apparently, restoration is a reactive strategy while protection is a proactive strategy. The restoration and protection solutions for SDN/OF networks often work as follows. The qualitative overview of those solutions are summarized in Table 6.

- *Data plane restoration* [71,72]: After the controller gets notification of a link failure, a list is made of all affected paths. For all these affected paths, a restoration path is calculated using a shortest path algorithm on the remaining topology. For affected switches which are on both the working and the restoration path, the flow entry is modified. For the other switches, there are 2 possibilities. If the switches are only on the failed path, the entries are deleted. If they are only on the restoration path, the new entries are added.
- *Data plane protection* [73,74]: In this operation, the protection path is pre-computed and it is installed together with the working path into the flow entries at switches, such that each switch has two forwarding information one for the protection path and the other for the

Table 6
Qualitative overview of different schemes of fault tolerance for data plane.

Fault tolerance for data plane			
Proposed approaches	Failure recovery schemes	Maximum restoration time	Maximum protection time
Fast failure recovery scheme [71,72]	Data plane restoration	(80—130 ms) > 50 ms	N/A
Carrier-grade recovery scheme [73]	Data plane restoration and protection	60 ms > 50 ms	(42—48 ms) < 50 ms
OpenFlow-based segment protection (OSP) scheme [74]	Data plane protection	N/A	64 ms > 50 ms

original working path. Once the failure is detected, e.g., via Bidirectional Forwarding Detection (BFD) [77], in the working path, the switch will use the protection path for flow forwarding.

Performance comparison of restoration and protection solutions: Compared with restoration solution that requires the deletion, modification, and addition operations between the controller and the switches during failures, the protection scheme can enable faster recovery without involving controller when failures are detected. Moreover, since protection mechanism is implemented at the switches by pre-installing the protection path, this would slightly increase the operations at flow setup time because extra protection information needs to be sent to the switches. However, in path protection, the bandwidth and latency requirements during failures can be significantly reduced because no interactions are required between switches and controller. For example, according to the experiments on a SDN network testbed with 14 switches [73], using the protection scheme, the maximum restoration time after failure detection is about 60 ms and all the flows are restored between 42 and 48 ms. This meets the 50 ms failure recovery time required by carrier-grade network. However, using restoration schemes, the failure recovery time can be in the range of 200—300 ms [71,72]. Therefore, for large-scale SDN systems, path protection solutions are more favorable in terms of achieving fast failure recovery.

5.1.2. Additional factors impacting fast failure recovery

Besides the centralized controller, the delay in failure recovery can be also caused by OF protocol. Specifically, according to OF specification, even if new flow entries are updated at the affected switch, the switch does not delete the entries using the failed link until the timeout of one of their associated timers, i.e., hard timer and soft timer, which is normally in the range of several seconds. This means that path failures are not actually recovered until one of the aforementioned timers expires. To encounter such problems, in [71,72], the protection or backup paths are pre-computed and installed using the GroupTable functionality of OF specification v1.1 as the "fast failover" mode. Once the failure in the working path is identified, the action bucket associated with this path in the GroupTable is made unavailable immediately by changing the value of its alive status. As a consequence, the packet arriving at the switch will be treated according to the next available bucket associated with the protection path. Instead of using GroupTable, OF-based segment protection (OSP) scheme [74] employs flow entry priority and auto-reject mechanism to realize fast switch-over between working path and protection path. More specifically, by assigning high priority to working path entries and low priority to protection path entries, it is guaranteed that all flows are forwarded via the working path if no failures are detected. Upon failures detected, the auto-reject mechanism allows all affected flow entries using the failed links to be deleted immediately without waiting for the soft or hard timeout. By this way, the affected flows can be timely restored and deviated by the switches so that they can research their destinations using the protection paths.

To achieve fast failure recovery, it is also demanding to inform the switches affected by the link failures as soon as possible. Moreover, this can effectively avoid waste of bandwidth by stopping the relevant switches from sending messages towards the direction of the failed links. Towards this, an algorithm is developed in [78], which allows switches to exchange simple link failure messages (LFMs) in such a way that the relevant switches can be aware of a link failure in a much shorter time than what it takes for controller to identify a link failure and send out the topology update. The advantage of this algorithm is that it does not involve controller, while no control message needed to be flooded in the entire network. However, the performance of this algorithm depends on the number of switches, i.e., if a network has a lot of switches that send flows towards the failed link, it will take longer to send LFM to all of them, and also depend on the total number of flow table entries in a switch, i.e., the larger the number of flow table entries, the longer it takes to search for the flows that go towards the failed link.

5.2. Fault tolerance for control plane

Because SDN is a logical centralized architecture, which relies on the controller to update policies and take actions when new flows are introduced in the network, reliability of the control plane is of critical importance. Without resolving a single point failure in the control plane, the entire network may be negatively recovered. The most fundamental mechanism to recover control plane failures in the centralized network is the "primary-backup replication" approach, where backup controllers will resume the network control in the case of primary controllers failures [79]. Two problems have to be addressed to support the replication schemes in SDN.

- *Coordination protocols between primary and backup controllers:* The OF protocol provides the possibility to configure one or more backup controllers, but OF does

not provide any coordination mechanism between the primary controller and the backups. Thus, the coordination protocols are needed, which are capable of performing the coordination between controllers to keep the backup consistent with the primary, which will turn the network to a safe state with minimal overhead imposed on hosts and switches [80].

- *Backup controller deployment:* The problem of placing controllers in SDNs aims to maximize the reliability of control networks. Specifically, (1) the impact of the controller number on reliability needs to be determined, and (2) the tradeoffs between reliability and latencies should be considered [81,68].

5.2.1. Primary and backup controller coordination

A new component called CPRecovery, which runs independently on top of the network OS, is developed in [80] to support the primary-backup mechanism. By CPRecovery, the replication process between the switch component running on the primary controller and the secondary controller works as follows: the switch sends an inactivity probe with an amount of a waiting time setup via the connection with the controller. If the controller does not send a reply within the waiting time, the switch assumes that the controller is down. In the recovery phase, the CPRecovery component acts during a failure state of the primary controller; the switch searches for the next network OS (the secondary controller acting as a backup) in its list and starts a connection to it. If the secondary controller receives a connection request from the switch, it generates a data path join event and changes its internal state to primary controller (become a primary controller). The new current primary keeps trying to send the state update messages to the former primary controller (become a secondary controller). For the experimental results, the response time is evaluated with different replication degrees in [80]. (The response time includes that the controller takes to process a request from the switch, sends a state update message to the backup controllers, receives a confirmation and sends a confirmation to the switch.) Four controllers are used in their experiments, one for the primary, three for the backup controllers (the replication degree = 3). The average response time without any secondary controller is around 8 ms, and the average response time with the replication degree = 1 (such as one secondary controller added) is around 14 ms. The 75% increase of the average response time is because it takes time to send and receive a confirmation from the secondary controller. Thus, the increased response time depends on the increased replication degree.

5.2.2. Backup controller deployment

In [68], the impact of the number of controllers on latency and reliability is analyzed, based on the average and worst-case propagation latencies on real topologies using the Internet2 OS3E [82]. From the analysis results of the Internet2 OS3E, it is shown that the average latency of one controller may be reduced to half by three controllers, while the same reduction for worst-case latency requires 4 controllers. Hence, the latency is deceased by $1/k$ controller for the single-controller latency. A $(3+1)$ controller setup is suggested in [68] for three load-balancing controllers and one backup controller for fault tolerance in the SDN architecture. Moreover, it is shown that one controller with a 10 ms latency (as a response time between the controller and the switches when a new packet flow is requested) is enough to meet a specified latency bound given the network size is between 8 and 200 nodes.

The problem of placing controllers is further addressed in [81] by investigating the impact of controller number on the tradeoffs between reliability and latency. The simulations are based on real topologies using the Internet2 OS3E [82] (a testbed of the multiple controllers for SDN) and Rocketfuel [83] (to analyze the actual values at the router-level maps of the ISP network). The failure probabilities of each switch and each link are set with the 0.01 and 0.02, respectively. According to the experiments, the best controller number is between $[0.035n, 0.117n]$ (where n is the total number of network nodes) as the minimum number of controllers is 3 and the maximum number of controllers is assumed to be 11. To determine the tradeoffs between the reliability and latency, it is suggested that the best controller placement is using one controller that yields the optimal reliability metric, while optimizing the average latency. However, when 3–4 controllers are placed, optimizing average and worst-case latency decrease the reliability. On the other hand, optimizing reliability increases the average and worst-case latency.

5.2.3. Distributed controller clusters in equal mode with a logical central view

Based on the possible deployment of SDN/OF controllers in equal mode as introduced in OF v1.2 [1], SOX [48] takes the approach of a centralized controller clusters while many controllers could be concurrently run in equal mode and the cluster shares a common Network Information Base (NIB). Such an architecture enabled automatic fail-over and load balancing, while the number of controller instanced would increase or decrease dynamically by adapting to the changing traffic demands. It is first demonstrated in the ONF PlugFest in October 2012 [48] and showcased in the first SDN World Congress [84].

Later a distributed SOX (DSOX) [85] is designed in which each centralized cluster is aimed to serve a large metropolitan area, or a particular autonomous system (AS). It utilizes a centralized NIB with the required information for globally optimized routing and resource scheduling, which has a globally centralized view and control over all its distributed domains. It should be noted that the designers of DSOX intentionally tried to limit the amount of data synchronization for consistency, and the newly updated domain network states or key traffic statistics would either be updated periodically or triggered by some special events.

5.3. Open research issues

Although some fault tolerance mechanisms are proposed at both data and control planes, there are still many open research problems to achieve a high reliability in SDN networks:

- *Fast and cost-efficient failure recovery for data plane:* From the research contributions discussed above, it is clear that the protection mechanism is the most appropriate approach for a high reliability performance with low-overhead communications between the controller and the switches in SDNs. However, this mechanism consumes high memory resources because the protection forwarding table may be installed on TCAM at each switch. Moreover, if the network policy is changed, then the pre-installed protection forwarding table should be updated too by following the new network policy, which can produce additional communication and operation overhead between the controller and the switches in the SDN network. Therefore, fast failure recovery mechanisms should be implemented in such a way that the fast failure recovery can be achieved with low communication overhead, no/less interference to SDN controller, and with the minimum intelligence available at the switches.
- *Traffic-adaptive primary-backup replication for control plane:* The centralized control plane has the critical reliability issue, such as a single point failure. To solve this problem, the primary-backup replication approach is commonly used in the centralized network. However, there are several open problems regarding how to define an optimal number of controllers and the best locations of the controllers for the primary control and the backup control(s) with an optimal tradeoff between reliability and latencies for time-varying traffic patterns, such as traffic volume trends in the entire network. These challenging issues should be accounted into the implementation of the fault tolerance mechanism in order to achieve a high reliability and an optimal performance of SDN controller(s).

6. Topology update

In this section, we focus on the planned changes (such as the network policy rules changes), instead of unplanned events (such as the network element/link failures) [86]. The general update operations are implemented as follows: each packet or flow is identified when updating the network from old policy to the new policy over multiple switches, and then each individual packet or flow is guaranteed to be handled by either the old policy or the new policy, but not by the combination of the two [86]. There are two types of consistency.

- *Per-packet consistency:* means that each packet flowing through the network will be processed according to a single network configuration.
- *Per-flow consistency:* means that all packets in the same flow will be handled by the same version of the policy. Hence, the per-flow abstraction preserves all path properties. These properties are expressed by the sets of packets belonging to the same flow that go through the network.

The key challenges are how SDN controller efficiently updates the network with consistency and also in real time.

6.1. Duplicate table entries in switches

To implement a consistent update for per-packet, a simple generic idea is proposed in [86,87], where the controller installs new configuration rules on all of the switches in a header field with the new version number; the ingress switches mark their new policy with the version number to incoming packets; meanwhile other switches can process packet with either the old or new policy, depending on the version number on the packet, but any individual packet is handled by only one policy; once all packets following the old policy have left the network, the controller deletes the old configuration rules from all switches, and then the update configuration is done. The efficiency of this algorithm depends on the explicit information on how long the switches need to hold the old rules because the limited memory, particularly the Ternary Content Addressable Memory (TCAM) at switches is not sufficient to hold a large-size forwarding tables configured by both old and new rules. Per-flow consistency guarantees that all packets in the same flow will be handled by the same version of the policy as long as its rule imposed on the flow does not time out. In particular, when the controller installs the new configuration, it sets a timeout for the old configuration rule. During this period, the incoming flows are handled by the old version until the rule expires. However, this algorithm only considers the scenario where multiple flows are processed using the same rule while leaving problem of handle flows with different rules open.

The key problem of above duplicated table entries scheme is that it requires holding both old and new sets of rules on the network switches. In the worst case, the switches holding the both policy rules can have a 100% overhead in terms of rule space resource consumption on the switches. To solve this problem, a more efficient update algorithm is introduced in [88] by adding a transfer function f^s at each switch s to perform policy rule replacement from *old* to *new* with high flow initiation data rate between the controller and the switches. A similar work can be found in [89], which introduces a generic update algorithm for implementing consistent update that considers a tradeoff between update time and rule-space overheads as in [86,87].

6.2. Time-based configuration

Time-based configuration method is introduced in [90,91] to allow coordinated SDN network updates in multiple devices, such that the controller can invoke a coordinated configuration change by sending update messages to multiple switches within either the same scheduled execution time or the different scheduled time, based on a time-based sequence of different update times. This approach was designed to simplify complex update procedures and to minimize transient effects caused by configuration changes. The implementation is very simple. For example, the controller sends a new updated policy with different time-based updates to each switch in such a way that switch 1 is scheduled to update its configuration with the new policy at time t, switch 2 at $t+$, and so on. The big problem of this solution is that in OF networks,

Table 7
Qualitative overview of different schemes of monitoring framework.

Monitoring tools			
Proposed approaches	Description	Traffic engineering technology	Analysis
PayLess [34]	Query-based monitoring	• Adaptive polling based on a variable frequency flow statistics collection algorithm.	• High accuracy and high overhead within a short minimum polling interval. • Low accuracy and low overhead within a large minimum polling interval.
OpenTM [94]	Query-based monitoring	• Periodically polling the switch on each active flow for collecting flow-level statistics.	• High accuracy and high overhead.
FlowSense [95]	Passive push-based monitoring	• Using the PacketIn and FlowRemoved messages in OpenFlow networks to estimate per flow link utilization for reducing monitoring overhead.	• High accuracy and low overhead compared with the Polling method.
OpenSketch [96]	Query-based monitoring	• Using the wildcard rules at switches for only monitoring a large aggregate of flows instead of all flows for reducing monitoring overhead. • Using a hierarchical heavy hitter algorithm for achieving high accuracy.	• Low memory consumption with high accuracy.
MicroTE [37]	Push-based monitoring	• Implemented on a server machine separated from the controller machine. • Advantages: (a) allows MicroTE to proactively respond when traffic demands change significantly, (b) reduces the processing overhead at the controller for collecting flow statistics, and (c) allows MicroTE to scale to a large network.	• Low consumed network utilization.
OpenSample [97]	Push-based monitoring	• Using the packet sampling tool sFlow [36] and TCP sequence numbers for achieving low latency. • Enabling traffic engineering to fast detect elephant flows and estimate link utilization of every switch port.	• Low latency measurement with high accuracy for both network load and elephant flows.

the controller must wait for an acknowledgment from one switch for completing the update before sending the update new policy to other switch until the network is completely updated.

A real-time network policy checking approach called Net-Plumber, is proposed in [92], which is based on Header Space Analysis (HSA) [93], and is able to configure the forwarding table with significantly fast update time. The NetPlumber Agent sits between the control plane and the switches, and it uses the HSA algorithm that can be used to check a rule update against a single policy within 50—500 μs. Instead of updating all the switches simultaneously, it incrementally updates only the portions of the switches affected by the changing rules in the network by using the plumbing graph that caches all possible paths of flows over the network to quickly update the reachable switches of a path for the flow, which is filtered by the OF rule (e.g., match, action). By this approach, it can update the network policy changes in real-time.

6.3. Open research issues

A consistency of topology update schemes may be considered in two different network scenarios.

- *A single controller in the SDN network:* How can the SDN controller efficiently update the network information with consistency in real-time without packet losses?
- *Multiple controllers in the multi-domain SDN networks:* If there are multiple SDN controllers in the large-scale or wide-area region network, then how can they consistently update the shared network information in the entire network with the tradeoff between the low inter-synchronization overhead and the real-time update?

7. SDN traffic analysis

In this section we discuss current network monitoring tools for network management, network verification and debugging in SDN architectures. The qualitative overview of the different monitoring solutions is summarized in Table 7.

7.1. Monitoring framework

Monitoring is crucial for network management. The management applications require accurate and timely statistics on network resources at different aggregation levels (such as flow, packet and port) [34]. The flow-based programmable networks, such as SDNs, must continuously monitor performance metrics, such as link utilization, in order to quickly adapt forwarding rules in response to changes in workload. However, existing monitoring solutions either require special instrumentation of the network or impose significant measurement overhead [95]. Many SDN architectures use the existing flow based network monitoring tools from traditional IP networks. For instance, the most prevalent one is NetFlow [35] from Cisco, which uses probe methods that are installed at switches as special modules to collect either complete or sampled traffic statistics, and send them to a central collector [82]. Another

flow sampling method is sFlow [36] from InMon, which uses time-based sampling for capturing traffic information. Another proprietary flow sampling method is JFlow [98], developed by the Juniper Networks. JFlow is quite similar to NetFlow. However, these approaches may be not efficient solutions to be applied in SDN systems, such as large-scale data center networks, because of the significantly increased overhead incurred by statistics collection from the whole network at the central controller. Therefore, the following solutions are seeking more efficient monitoring mechanisms in order to achieve both high accuracy and low overhead.

PayLess [34] is a query-based monitoring framework for SDN to provide a flexible RESTful API for flow statistics collection at different aggregation levels (such as flow, packet and port), where it performs highly accurate information gathering in real-time without incurring significant network overhead. To achieve this goal, instead of letting controller continuously polling switches, an adaptive scheduling algorithm for polling is proposed to achieve the same level of accuracy as continuous polling with much less communication overhead. Moreover, PayLess provides a high-level RESTful API, which can be accessed by any programming language. Therefore, it is very easy for different network applications to develop their own monitoring applications, and access the collected data from the PayLess data stored at different aggregation levels. The evaluation results show that PayLess has a very low overhead of only sending 6.6 monitoring messages per second on the average, compared with the controller's periodic polling, which has an overhead with 13.5 monitoring messages per second on average. The measurement of the trade-offs between accuracy and the monitoring overhead within the given *minimum polling interval* ($T_{\min}$) shows that the monitoring data is very accurate but the message overhead is very high for a short time interval, e.g., $T_{\min} = 250$ ms. However, at a large time interval, e.g., $T_{\min} = 2000$ ms, the message overhead is very low but the monitoring data error is very high. Thus, the monitoring accuracy increases at the cost of increased network overhead.

OpenTM [94] is a query-based monitoring method to estimate the traffic matrix (TM) for OF networks. OpenTMs logic is quite simple. It keeps tracking all the active flows in the network, gets the routing information from the OF controllers routing application, discovers flow paths, and periodically polls flow byte and packet-count counters from switches on the flow path. Using the routing information, OpenTM constructs the TM by adding up statistics for flows originated from the same source and destined to the same destination. It is similar to FlowSense to compute utilization with a low overhead in the OF network. However, by measuring network-wide traffic matrix by periodically polling one switch on each flow's path for collecting flow level statistics, it cause a significant overhead. By polling method that randomly selects some switches, it may affect accuracy if the switch is not carefully chosen.

FlowSense [95], in contrast to the on-demand approach used in OpenTM [94], is a passive push-based monitoring method to analyze control messages between the controller and switches. It uses the controller messages to monitor and measure network utilization such as the bandwidth consumed by flows traversing the link, without inducing additional overhead. For example, FlowSense uses the PacketIn and FlowRemoved messages in OF networks to estimate per flow link utilization. The evaluation results show that FlowSense has high accuracy compared with the Polling method and can accomplish 90% link utilization estimating jobs under 10 s based on a small testbed consisting of two OF switches and one controller.

OpenSketch [96] is a software defined traffic measurement architecture, which separates the measurement data plane from the control plane. OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting) at switches, which can be implemented with commodity switch components and support many measurement tasks. OpenSketch is both generic and efficient to allow more customized operations and thus can realize more efficient data collection with respect to choosing which flow to measure by using both hashing and wildcard rules. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resource for different measurement tasks. The three-stage pipeline is implemented on NetFPGA hard-ware as an OF switch. The OpenSketch library includes a list of sketches, the sketch manager, and the resource allocator. Sketches can be used for many measurement programs such as heavy hitters [99,100], the traffic change detection [101], the flow size distribution estimation [102], the global iceberg detection [103], and the fine-grained delay measurement [104]. Thus, OpenSketch makes measurement programming easier at the controller. The monitoring framework similar to the above solutions are also proposed in [105] where a monitoring frame-work utilizes secondary controllers to identify and monitor aggregate flows using a small set of rules that changes dynamically with traffic load. This framework monitors only a large aggregate of flows instead of monitoring all flows as PayLess [34]. This monitoring framework is based on wildcard rules (at switches) that match one bit in the packet header, and includes a hierarchical heavy hitter (HHH) algorithm [100] in order to achieve high accuracy with low monitoring overhead. A framework is proposed in [106], which can instruct hash-based switches for collecting traffic information, along with the HHH algorithm for defining important traffic, to support different measurement tasks with trade-offs between accuracy and overhead.

MicroTE [37] is a fine-grained traffic engineering scheme that works atop a variety of underlying data center network topologies. It has a monitoring component in the server, instead of letting the network controller periodically poll switches. Their solutions directly provide advantages allowing proactively respond to the changes in the traffic load, scale down a large network, and reduce the processing overhead imposed by the MicroTE on the network devices. This server-based system offers these advantages via the following approaches: (1) it allows the controller to receive triggered updates of traffic loads, especially when the traffic loads change significantly, while a purely switch based approach, at least in the current implementation of OF, only supports polling by the controller, which is far less flexible; (2) it prevents the

network controller from creating a significant amount of control traffic on the network by constantly polling all switches on nearly a per second granularity; and (3) it shifts the bottleneck of constantly generating flow statistics from the switches to the end hosts. Each of the servers in the monitoring components tracks the network traffic being sent/received over its interfaces as well as with whom these bytes were exchanged. However, only one server per rack is responsible for aggregating, processing, and summarizing the network statistics for the entire rack. This server, called the designated server, is also in charge of sending the summarized traffic matrix to the network controller. To fulfill its role, the designated server must be able to perform the following tasks: (1) collect data from other servers in the rack, (2) aggregate the server to server data into Rack to Rack data, (3) determine predictable ToR pairs (i.e., pairs of Top-of-Rack switches), and (4) communicate this information with the network controller.

OpenSample [97] proposed by IBM research lab, is a sampling-based SDN measurement system, which uses a packet sampling tool, sFlow [36], to capture packet header samples from the network with low overhead and uses TCP sequence numbers from the captured headers to measure accurate flow statistics. Using these two methods (packet samples and TCP sequence numbers) OpenSample extracts flow statistics for detecting elephant flows, estimating port utilization at each switch, generating a snapshot of the network state for use to other applications, and enabling traffic engineering. Thus, OpenSample achieves the low-latency measurements with high accuracy by using the sFlow with TCP sequence numbers rather than using the expensive OF rules, such that the counter function in OF switches for each flow table, flow entry, port, queue, group, group bucket, meter and meter band may be implemented in software and maintained by polling hardware mechanisms [1]. Moreover, the implementation of OpenSample does not need to modify the end-host server that is required by MicroTE [37].

7.2. Checking network invariants

The verification of the network invariants is an important task in SDN networks. SDN will simplify the development of network applications, but bugs are likely to remain problematic since the complexity of the software will increase [107]. Moreover, SDN allows multiple applications or even multiple users to program the same physical network simultaneously, potentially resulting in conflicting rules that alter the intended behavior of one or more applications [107,108].

VeriFlow [107] is a verification tool in order to achieve a real-time checking in SDN networks. It employs a similar concept of the real-time network policy checking from [93,92]. It is designed as a proxy residing between the controller and switches in the network for monitoring all communication in either direction and verifying network-wide invariant violations dynamically as each forwarding rule is inserted. The verification latency should be within a few milliseconds to achieve real-time responses according to [43] because the current SDN controllers are capable of handling around 30 K new flow installs per second while maintaining a sub-10 ms flow installation time. To implement VeriFlow with high speeds for every rule insertion or deletion in the forwarding table at each switch, VeriFlow slices the network into a set of equivalence classes of packets based on the destination IP address with a longest-prefix-match rule, which will only affect the forwarding of the packets destined for that prefix. By employing such approach, it is shown that network invariants can be verified within hundred of microseconds as new rules are installed into the network.

OFRewind [109] runs as a proxy on the substrate control channel between the controller and switches to enable recording and replay of events for troubleshooting problems in production networks.

FlowChecker [110] deploys the similar ideas in ConfigChecker [111] which uses Binary Decision Diagrams (BDDs) to analyze the end-to-end access control configuration of all network devices (such as routers, firewalls, IPSec gateways, NAT and multicasting), FlowChecker allows OF administrators/users to manually verify the consistency of multiple controllers and switches across different OF federated infrastructure. For example, it verifies a configuration of network rules such as forwarding rules in the forwarding tables by using Boolean expression to detect misconfigurations.

7.3. Debugging programming errors

Modern networks provide a variety of interrelated services including routing, traffic monitoring, load balancing, and access control. Unfortunately, the languages used to program today's networks lack some modern features. They are usually defined at the low level of abstraction supplied by the underlying hardware and they fail to provide even rudimentary support for modular programming. As a result, network programs tend to be complicated, error-prone, and difficult to maintain [112].

NICE [113] is an efficient and systematically technique tool, which is a combination of model checking and symbolic execution to efficiently discover violations of network-wide correctness properties due to bugs in the controller programs. By using NICE, the OF programmer can be instructed to check for generic correctness properties such as forwarding loops, black holes, or application-specific correctness properties, etc. The programming model checking tool relies on the controller program as a set of event handlers, a switch program as the values of all variables for defining the switch state and identifying transitions, and an end host program such as client/server or mobile user. NICE is implemented by using Python language to seamlessly support OF controller programs. Thus, NICE performs symbolic execution to explore all program code paths through an entire OF network.

ndb [114] is a debugging software tool, which allows SDN programmers and operators, to track down the root cause of a bug. The *ndb* network debugger inspired by *gdb* provides breakpoint and breaktrace keywords along with a packet backtrace function, which allows to define a packet breakpoint (e.g., an un-forwarded packet or a packet filter), and then shows the sequence of information relevant to code path, events, and inputs regarding a

forwarding packet. Thus, ndb can find bugs in any level in the SDN stack and it provides an idealized model better than NICE [113].

7.4. Open research issues

- *Traffic Analysis:* To achieve the potential benefits of the SDN-TE, there are still open challenges based on the following critical issues. The traffic analysis of the SDN-TE is significantly dependent on how global information related to application or traffic characteristics and states can be obtained in close-to real time fashion. Moreover, the global information can be obtained from 3G/4G cellular networks that have a tremendous growth of mobile data access and bandwidth usage. Thus, how to efficiently handle the bigdata with regard to user behavior, locality, and time-dependent statistics, is the major consideration in the developing SDN-TE. In [56], in-depth traffic pattern analysis method was presented as a bigdata analysis. According to [56], the bigdata analysis solution should include the parallel data mining method such as like the K-means clustering algorithm for analyzing a large volume of traffic data. Thus, the effective parallel data mining not only enables the extraction of various statistics, but also significantly speeds up the whole process. Such a combination of traffic analysis and data mining methods also makes it possible to derive more general conclusions about smartphone usage patterns.
- *Traffic monitoring:* The open challenge issues in traffic monitoring mechanisms are regarding how to reduce significant network overhead when SDN controller(s) or the monitoring device collects the network statistics with high accuracy.
- *Network invariant checking and programming error debugging methods:* The verification and debugging methods should work together with network security issues. How to early detect or prevent intrusions by using verification network or programming error checking methods is largely unexploited. The security mechanisms are out of scope areas of traffic engineering. Therefore, we did not account it in this survey studies.

8. Existing TE tools for SDN-OpenFlow networks

8.1. Industry solutions

Here we present the state of the art of TE tools for SDNs in the industry, which is summarized in Table 8.

B4 [115] designed by Google, is a Software Defined WAN for Googles data center networks. The centralized traffic engineering is applied to easily allocate bandwidth among competing services based on application priority, dynamically shifting communication patterns, and prevailing failure conditions. They address the critical performance and reliability issues that Wide Area Networks (WANs) faced when delivering terabits per second of

Table 8
Qualitative overview of existing industrial TE tools for SDN-OpenFlow networks.

Industry solutions			
Proposed approaches	Description	Traffic engineering technology	Analysis
B4 [115] from Google	A Software Defined WAN for Google's data center networks.	• Using the centralized Traffic Engineering (CTE) to adjudicate among competing resource demands, measure available network capacity for multi-path forwarding/tunneling, and dynamically reallocate bandwidth from the link or switch failures. • Using hash-based ECMP algorithm for load-balancing.	• Near 100% link utilization for the majority of the links and 70% link utilization overall.
SWAN [116] from Microsoft	A Software-driven WAN (SWAN) for inter-data center WANs.	• Using two types of sharing policies: (a) the different ranking classes of traffic, and (b) the same priority of traffic with the max–min fairness principle.	• 98% of the maximum allowed network throughput, compared with 60% in MPLS-enable WAN.
Dynamic routing for SDN [2] from Bell Labs	An optimized routing control algorithm for SDN.	• Using the Fully Polynomial Time Approximation Scheme (FPTAS) to solve the SDN controller optimization problem that minimize the maximum utilization of the links in the network.	• FPTAS based routing outperforms a standard OSPF routing in SDNs.
ADMCF-SNOS [84,49,50] from Huawei	An integrated resource control and management system for large centrally controlled or loosely coupled distributed network systems.	• Using the Adaptive Dynamic Multi-path Computation Framework (ADMCF) to provide the necessary infrastructure and algorithms for data collection, analysis, and various optimization algorithms. • Using Static and Dynamic Hybrid Routing (SDHR) algorithms for computing the optimal routing to provide a simpler and more resource efficient near-optimal hybrid routing than destination-bead or explicit routing schemes.	• SDHR based routing outperforms an explicit routing about 95% TCAM space saved at the normalized throughput about 70%.

aggregate bandwidth across thousands of individual links. Conventionally, WAN links can only achieve 30—40% average utilization and may also experience unexpected link failures. To encounter such problem, B4 is designed based on SDN principles and OF [1] to manage individual switches. The core part is the centralized Traffic Engineering (CTE), which allows adjustment among competing resource demands, measuring available network capacity for multi-path forwarding/tunneling, and dynamically reallocating bandwidth from the link/switch failures. The CTE architecture includes the network topology graph that represents sites as nodes, and site to site connectivity as edge switches. Using this graph, the aggregate traffic is computed at site–site edges, and then the abstract computed results are fed into the TE Optimization Algorithm for fairly allocating bandwidth among all Flow Group (FG), where FG can be generated for the individual application and represented by a tuple (including source site, destination site, QoS). The Tunnel (call) represents a site-level path in the networks, such as a sequence of connecting nodes for a path. The Tunnel Group (TG) maps FGs to a set of tunnels according to the weights that specify the fraction of FG traffic to be forwarded along each tunnel. For load-balance routing, B4 uses a hash-based ECMP algorithm. By employing the above schemes, B4 has managed to achieve high link utilization. For example, it is reported that in B4, all links have average 70% link utilization for long time periods (such as 24-h), while many links are running under 99% of the bandwidth utilization.

SWAN [116] is a Software-driven WAN (SWAN) proposed by Microsoft, which utilizes policy rules to allow inter-data center WANs to carry significantly more traffic for higher-priority services, while maintaining fairness among similar services. Conventionally, WAN is operated using MPLS TE based on ECMP routing, which can spread traffic across a number of tunnels between ingress-egress router pairs. However, this approach yields very low efficiency due to the lack of global view at edge router/switches. In this case, greedy resource allocation has to be performed for a flow by using the shortest path with available capacity (CSPF). To solve the above problems, SWAN exploits the global network view enabled by the SDN paradigm to optimize the network sharing polices, which allows WAN to carry more traffic and support flexible network-wide sharing. More specifically, two types of sharing policies are employed. First, a small number of traffic classes, e.g., interactive traffic, elastic traffic, and background traffic, are ranked according to their priorities, and then the network resources are allocated among the traffic flows based on their priorities. Second, the traffic flows with the same priority are allocated with the network resource according to the max–min fairness principle. As a consequence, SWAN carries about 98% of the maximum allowed network traffic, while MPLS-enable WAN only carries around 60%.

Dynamic routing for SDN [2] proposed at Bell Labs, addresses the routing optimization problem using the Fully Polynomial Time Approximation Scheme (FPTAS). The optimization problem aims to find the optimal routes for network traffic flows such that delay and packet loss at the links are minimized, thus define how to minimize the maximum utilization of the links in the network. Specifically, FPTAS in [2] solves the dynamic routing problem instead of a standard linear programming problem. FPTAS is very simple to implement and runs significantly faster than a general linear programming solver. The algorithms are implemented as a SDN routing on ns-2 simulator [117]. Compared with a standard OSPF routing, it shows that the proposed SDN routing outperforms OSPF routing in terms of overall network throughput, delay and packet loss rate.

ADMCF-SNOS [49,50], the Adaptive Dynamic Multipath Computation Framework for Smart Network Operating Systems (ADMCF-SNOS) from Shannon Lab of Huawei utilizes its Smart Network Operating System (SNOS) is to provide an integrated resource control and management system for large centrally controlled or loosely coupled distributed network systems. The management applications are built on top of the Smart OF Controller (SOX)[48] enhanced by dynamic resource-oriented APIs. One of such an application was the Adaptive Dynamic Multi-path Computation Framework (ADMCF). ADMCF [49] was designed as an open and easily extensible solution framework that can provide the necessary infrastructure and algorithms for data collection, analysis, and various optimization algorithms. The ADMCF designers believed that it would be impossible for any single optimization algorithm to get satisfactory solutions for a large centrally controlled network while its topology, states, and more critically application traffic can change rapidly. Instead, a set of algorithms that work together in an adaptive and intelligent fashion would be more capable to provide a more adequate global routing and resource allocation optimization. As it would be costly for the central optimization algorithms to calculate good routes dynamically, such a framework should take advantage of many hidden patterns in the combinations of network topology, states, and traffic flows.

ADMCF consists of four main components: (1) *Routing Policy* & *Rule Configuration* – Administrator or Network OS specify and configure various policies and rules based on global network information, client/application QoS requirements, traffic statistics and Patterns, etc. (2) *Adaptive* & *Dynamic Multi-Path Computation* – Innovative combination of enhanced edge-disjoint path algorithms with iterative CSPF algorithm, and/or other heuristics that can truly perform global optimization. (3) *Path Evaluator/Assessor* – a mechanism that can take into account of contributing factors in the evaluation and selection of paths obtained from above set of algorithms. (4) *Path DB* – selecting proper paths and update the Path DB.

Static and Dynamic Hybrid Routing (SDHR) [118] [119] Classical TE methods calculate the optimal routing based on a known traffic matrix. However, it is very difficult to get an accurate traffic matrix in a large operational network because of frequent changes of service demands. Thus, it is of interest to find a set of good routing configuration to accommodate for a wide range of traffic matrices and offers the near optimality of performance for each such traffic matrix. SDHR intended to provide a simpler and more resource efficient near-optimal hybrid routing solution than destination-based or explicit routing. For any

Table 9
Qualitative overview of existing academic TE tools for SDN-OpenFlow networks.

Academic solutions		
Proposed approaches	Description	Traffic engineering technology
Plug-n-Serve [120] from Stanford University	An OpenFlow enabled web server load-balancing application.	Uses the LOBUS algorithm for flow management so it can add/delete servers to unstructured network for traffic adjustments.
Aster*x [121] from Stanford University	An OpenFlow enabled load-balancing application.	Enhanced version of the Plug-n-Serve to manage a large network of switches and servers for minimizing the average response time of web services.
OpenFlow-based load balancer [42] from Princeton University	An OpenFlow enabled load-balancing application.	Using wildcard rules, switches can handle "microflows" without involving the controller.
FlowVisor [122] from Stanford University	A network virtualization tool.	It is a proxy protocol to sit between the multiple controllers and the switches in order to allow multiple controllers to share the same network infrastructure without interfering with each other.

specific traffic demands, it would adapt to some "best" suited routing decisions. Its hybrid routing achieves load balancing for multiple traffic matrices, by complementing destination-based routing with a small number of explicit routing decisions to take advantage of both approaches. Hybrid routing greatly reduces the number of forwarding entries and thus requires less TCAM resources. For the four test networks frequently used for such comparison and the two 500 nodes randomly generated subnetworks, SDHR demonstrated near-optimal load balancing improvements on "normalized throughput" from 35% to over 70%, while saving up to 95% TCAM resources compared to explicit routing. The approach is to pre-compute a basic destination-based routing and multiple sets of complementary explicit routing, and then dynamically apply different set of explicit routing to achieve load balancing for a wide range of traffic matrices, according to traffic changes.

The OF-enabled or compliant switches can easily support such combination of destination-based routing and explicit routing in their forwarding tables with the centralized controller. The controller can install both destination-based routing entries and multiple sets of explicit routing entries in the flow tables, while at any given time, only the set of explicit routing that "best matches" the current traffic patterns is active. In hybrid routing, if a packet matches both active explicit routing and destination-based routing entries, the active explicit routing entry will take precedence to forward the packet. The multi-path forwarding with ADMCF and its algorithms were successfully tested at EANTC [84].

8.2. Academic solutions

Here we present the TE tools for SDNs in the academia, which are summarized in Table 9.

Plug-n-Serve [120] developed at Stanford University, is an OF-enabled load balancing application for web traffic. It tries to minimize response time by controlling the load on the network and the servers using customized flow routing. It operates in an unstructured network topology. Plug-n-Serve can add new servers to unstructured network, detect the changes and make traffic adjustments that minimize the response time. Plug-n-Serve uses LOBUS (LOad-Balancing over UnStructured networks) algorithm for Flow Manager which is effective in adding servers to network such that LOBUS automatically expands its view of the network and appropriately shares the load over the added devices.

Aster*x[121] also developed at Stanford University is an improved version of Plug-n-Serve [120] and can be used at a much larger scale network. Aster*x is a server load-balancing system that effectively minimize the average response time of web services in unstructured networks built with cheap commodity hardware. Using OF to keep track of state and to control the routes allows the system to be easily reconfigured; the network operator, thus, can add or remove capacity by turning hosts on or off, and add or remove path diversity by turning switches on or off. In addition, the system allows operators to increase the capacity of the web service by simply plugging in computing resources and switches in an arbitrary manner. Aser*x load-balancing system has three functional units: *Flow Manager* for the OF controller that manages and routes flows based on the specific load-balancing algorithm chosen. *Net Manager* probes the network and keeps track of the network topology and its utilization levels. It queries switches periodically to get link usage and monitor the latency experienced by packets traversing the links. *Host Manager* monitors the state and load at individual servers in the system, and reports the collected information to the Flow Manager. By employing above schemes, the SDN controller of Aser*x system is capable of managing a large network of switches and servers.

OpenFlow-based load balancer [42] developed at Princeton University, is an efficient load-balancing architecture, which includes the partitioning algorithm for generating wildcard rules that are proactively installed into the switches to direct requests for "microflows" without involving the controller. More specifically, by this load-balancing approach, the switch performs an "action" of rewriting the server IP address and forwarding the packet to the output port associated with the chosen replica server by using the wildcard rules.

FlowVisor [122] originally developed at Stanford University and continued in ON.LAB [123]. FlowVisor is a network virtualization application which can be considered as a proxy protocol to sit logically between the multiple controllers and the OF switches in order to allow multiple controllers to share the same network infrastructure without interfering with each other. Since the main purpose of FlowVisor is to provide virtualization in OF networks, it does not provide many traffic engineering

mechanisms. In particular, it can allocate the link bandwidth by assigning a minimum data rate to the set of flows that make up a slice, and also divides the flow-table in each switch by keeping track of which flow-entries belong to which controller. For handling new flow messages, when a packet arrives at a switch that does not match an entry in the flow table, a new flow message is sent to the controller. When there are multiple controllers, the new flow requests may occur too frequently. To process these flows on a switch with limited TCMA memory, the FlowVisor tracks the new flow messages arrival rate for each slice, and if it exceeds given threshold, the FlowVisor will insert a forwarding rule to drop the problem packets for a short period. Therefore, the FlowVisor needs a specific TE for supporting significant flows for a special controller among the multiple controllers to define routes depending on the different priority of traffic.

9. Conclusions

In this paper, we provide an overview of traffic engineering mechanisms in SDN architectures. We study the traditional traffic engineering technologies from early ideas in ATM networking through current developments in IP and MPLS networking. In particular, we investigate the traffic management with regard to load balancing, fault tolerance, consistent network update methods, as well as traffic analysis for testing and debugging network systems and network monitoring tools. We cover important network aspects of availability, scalability, reliability, and consistency in data networking with SDN. Moreover, we study the traffic engineering mechanisms and describe how to apply them to SDN/OF networks. SDN is a fast-evolving research area in data networking with open research issues. For availability and scalability issues, SDN-TE system should manage data flow efficiently at both the control plane and the data plane with the tradeoffs between latency and load-balance. For reliability issues, in the data plane, fast failure recovery mechanisms should be implemented with low-overhead communications between the controller and the switches. In the control plane, the fault tolerance mechanisms must consider a single point failure and should define an optimal number of controllers and the best location of controllers for the primary control and the backup controller(s) with a tradeoff between reliability and latencies of a variety of traffic patterns in the entire network. For consistency issues, the SDN controller efficiently updates the network with consistency in real-time and safety without packet drops, and with low synchronization overhead. Thus, SDN's effectiveness and great potential for next generation data networking come with many new technical challenges, which need to be addressed by the new research advances.

Acknowledgment

The authors would like to thank Caterina Scoglio, Mehmet Can Vuran, Eylem Ekici, and Xudong Wang, for their valuable comments and suggestions to improve the quality of the paper.

References

[1] Openflow switch specification v1.0–v1.4 <https://www.opennetworking.org/sdn-resources/onf-specifications>.
[2] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: Proceedings of the 32nd IEEE International Conference on Computer Communications, INFOCOM'13, April 2013, pp. 2211–2219.
[3] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao, Overview and Principles of Internet Traffic Engineering, RFC 3272, Tech. Rep., May 2002.
[4] J.J. Bae, T. Suda, Survey of traffic control schemes and protocols in ATM networks, Proc. IEEE 79 (2) (1991) 170–189.
[5] S. Akhtar, Congestion Control in a Fast Packet Switching Network, Ph.D. dissertation, Ph.D. dissertation, Department of Electrical Engineering, Washington University, 1987.
[6] M. Hirano, N. Watanabe, Characteristics of a cell multiplexer for bursty ATM traffic, in: Proceedings of IEEE International Conference on Communications, ICC'89, June 1989, pp. 399–403.
[7] S. Jacobsen, K. Moth, L. Dittmann, K. Sallberg, Load control in atm networks, in: Proceedings of the 8th International Switching Symposium, vol. 5, 1990, pp. 131–138 (Annual report).
[8] T. Kamitake, T. Suda, Evaluation of an admission control scheme for an atm network considering fluctuations in cell loss rate, in: Proceedings of Global Telecommunications Conference, GLOBECOM'89, November 1989, pp. 1774–1780.
[9] I. Cidon, I.S. Gopal, "Paris: An approach to integrated high-speed private networks, Int. J. Dig. Anal. Cab. Syst. 1 (2) (1988) 77–85.
[10] G. Gallassi, G. Rigolio, L. Fratta, ATM: bandwidth assignment and bandwidth enforcement policies, in: Proceedings of Global Telecommunications Conference, GLOBECOM'89, November 1989, pp. 1788–1793.
[11] R. Chipalkatti, J. Jurose, D. Towsley, Scheduling policies for real-time and non-real-time traffic in a statistical multiplexer, in: Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies, Technology: Emerging or Converging, INFOCOM'89, vol. 3, April 1989, pp. 774–783.
[12] P. Yegani, M. Krunz, H. Hughes, Congestion control schemes in prioritized ATM networks, in: Proceedings of IEEE International Conference on Communications, ICC'94, Serving Humanity Through Communications, SUPERCOMM/ICC'94, May 1994, pp. 1169–1173.
[13] S. Kamolphiwong, A. Karbowiak, H. Mehrpour, Flow control in atm networks: a survey, Comp. Commun. 21 (11) (1998) 951–968.
[14] N. Wang, K. Ho, G. Pavlou, M. Howarth, An overview of routing optimization for internet traffic engineering, Commun. Surv. Tut., IEEE 10 (1) (2008) 36–56 (First Quarter).
[15] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, C. Diot, Analysis of link failures in an IP backbone, in: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement, November 2002, pp. 237–242.
[16] B. Fortz, M. Thorup, Internet traffic engineering by optimizing OSPF weights, in: Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM'00, vol. 2, March 2000, pp. 519–528.
[17] B. Fortz, J. Rexford, M. Thorup, Traffic engineering with traditional ip routing protocols, IEEE Commun. Magaz. 40 (10) (2002) 118–124.
[18] B. Fortz, M. Thorup, Optimizing OSPF/IS–IS weights in a changing world, IEEE J. Select. Areas Commun. 20 (4) (2006) 756–767.
[19] N. Deo, C.-Y. Pang, Shortest-path algorithms: taxonomy and annotation, Networks 14 (2) (1984) 275–323.
[20] C.E. Hopps, Analysis of an Equal-Cost Multi-Path Algorithm, RFC 2992, Tech. Rep., November 2000.
[21] G. Rétvári, T. Cinkler, Practical OSPF traffic engineering, IEEE Commun. Lett. 8 (11) (2004) 689–691.
[22] S. Kandula, D. Katabi, S. Sinha, A. Berger, Dynamic load balancing without packet reordering, ACM SIGCOMM Comp. Commun. Rev. 37 (2) (2007) 51–62.
[23] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in: Proceedings of Networked Systems Design and Implementation Symposium, NSDI'10, vol. 10, April 2010, pp. 19–19.
[24] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4d approach to network control and management, ACM SIGCOMM Comp. Commun. Rev. 35 (5) (2005) 41–54.
[25] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, Design and implementation of a routing control platform, in: Proceedings of the 2nd Conference on Symposium on

Networked Systems Design & Implementation, NSDI'05, May 2005, pp. 15–28.
[26] D.O. Awduche, J. Agogbua, Requirements for Traffic Engineering Over MPLS, RFC 2702, Tech. Rep., September 1999.
[27] D.O. Awduche, MPLS and traffic engineering in IP networks, IEEE Commun. Magaz. 37 (12) (1999) 42–47.
[28] I.F. Akyildiz, T. Anjali, L. Chen, J.C. de Oliveira, C. Scoglio, A. Sciuto, J.A. Smith, G. Uhl, A new traffic engineering manager for diffserv/MPLS networks: design and implementation on an IP QoS testbed, Comp. Commun. 26 (4) (2003) 388–403.
[29] G. Swallow, MPLS advantages for traffic engineering, IEEE Commun. Magaz. 37 (12) (1999) 54–57.
[30] A.R. Sharafat, S. Das, G. Parulkar, N. McKeown, MPLS-TE and MPLS VPNS with openflow, ACM SIGCOMM Comp. Commun. Rev. 41 (4) (2011) 452–453.
[31] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, H. Green, Openflow MPLS and the open source label switched router, in: Proceedings of the 23rd International Teletraffic Congress, ITC'11, September 2011, pp. 8–14.
[32] B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, S. Ueno, Requirements of an MPLS Transport Profile, RFC 5654, Tech. Rep., September 2009.
[33] A.R. Curtis, W. Kim, P. Yalagandula, Mahout: low-overhead datacenter traffic management using end-host-based elephant detection, April 2011, pp. 1629–1637.
[34] S.R. Chowdhury, M.F. Bari, R. Ahmed, R. Boutaba, Payless: a low cost network monitoring framework for software defined networks, in: Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium, NOMS'14, May 2014.
[35] Netflow <http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.html>.
[36] sflow <http://www.sflow.org/sFlowOverview.pdf>.
[37] T. Benson, A. Anand, A. Akella, M. Zhang, Microte: fine grained traffic engineering for data centers, in: Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies, CoNEXT'11, December 2011, p. 8.
[38] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Comp. Commun. Rev. 38 (2) (2008) 69–74.
[39] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: scaling flow management for high-performance networks, ACM SIGCOMM Comp. Commun. Rev. 41 (4) (2011) 254–265.
[40] J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A.R. Curtis, S. Banerjee, Devoflow: cost-effective flow management for high performance enterprise networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, HotNets-IX, October 2010, p. 1.
[41] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with difane, ACM SIGCOMM Comp. Commun. Rev. 40 (4) (2010) 351–362.
[42] R. Wang, D. Butnariu, J. Rexford, Openflow-based server load balancing gone wild, in: Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE'11, March 2011, pp. 12–12.
[43] A. Tavakoli, M. Casado, T. Koponen, S. Shenker, Applying nox to the datacenter, in: Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), October 2009.
[44] A. Tootoonchian, Y. Ganjali, Hyperflow: a distributed control plane for openflow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, INM/WREN'10, April 2010, p. 3.
[45] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al., Onix: a distributed control platform for large-scale production networks, in:' Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, vol. 10, October 2010, pp. 1–6.
[46] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, Balanceflow: controller load balancing for openflow networks, in: Proceedings of IEEE 2nd International Conference on Cloud Computing and Intelligent Systems, CCIS'12, vol. 2, October 2012, pp. 780–785.
[47] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, August 2012, pp. 19–24.
[48] M. Luo, Y. Tian, Q. Li, J. Wang, W. Chou, Sox – a generalized and extensible smart network openflow controller (x), in: Proceedings of the First SDN World Congress, Damsdadt, Germany, October 2012.
[49] M. Luo, Y. Zeng, J. Li, An adaptive multi-path computation framework for centrally controlled networks, 2014, submitted for publication..
[50] M. Luo, X. Wu, Y. Zeng, J. Li, In-memory fast multi-dimensional methods for network information storage and query in sdn-openflow networks, CoNext'14 (2014) submitted for publication.
[51] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M.F. Kaashoek, R. Morris, Flexible, wide-area storage for distributed systems with wheelfs, in: Proceedings of the 6th USENIX symposium on Networked Systems Design and Implementation, NSDI'09, vol. 9, April 2009, pp. 43–58.
[52] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, Nox: towards an operating system for networks, ACM SIGCOMM Comp. Commun. Rev. 38 (3) (2008) 105–110.
[53] E. Ng, Maestro: A System For Scalable Openflow Control, Rice University Technical Report TR10-08, December 2010.
[54] D. Erickson, The Beacon Openflow Controller, 2012 <https://openflow.stanford.edu/display/Beacon/Home>.
[55] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE'12, April 2012.
[56] L. Qian, B. Wu, R. Zhang, W. Zhang, M. Luo, Characterization of 3g data-plane traffic and application towards centralized control and management for software defined networking, in: Proceedings of IEEE International Congress on Big Data (BigData Congress'13), June 27 2013–July 2 2013, pp. 278–285.
[57] Enatc.de <http://www.eantc.de/fileadmin/eantc/downloads/events/2011-5/MPLSEWC2013/EANTC-MPLSEWC2013-WhitePaper-5.1.pdf>.
[58] Openflow switch specication <http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf>.
[59] Huawei <http://enterprise.huawei.com/ilink/enenterprise/download/HW_308596>.
[60] Hp <http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA4-6562ENW.pdf/>, <http://h17007.www1.hp.com/us/en/networking/products/switches/HP_5900_Switch_Series/>.
[61] Nec <http://www.necam.com/SDN/>, <http://www.necam.com/sdn/doc.cfm?t=PFlowPF5240Switch/>, <http://www.necam.com/docs/?id=5ce9b8d9-e3f3-41de-a5c2-6bd7c9b37246>.
[62] Ibm <http://www.redbooks.ibm.com/technotes/tips0815.pdf/>, <http://www-03.ibm.com/systems/networking/switches/rack/g8264/features.html>.
[63] Pica8 <http://www.pica8.com/open-switching/1gbe-10gbe-40gbe-open-switches.php>.
[64] Broadcom <http://www.broadcom.com/collateral/pb/OF-DPA-PB100-R.pdf>.
[65] Brocade <http://www.brocade.com/products/all/routers/product-details/netiron-mlx-series/features.page/>, <http://finance.yahoo.com/news/brocade-advances-sdn-leadership-openflow-130200421.html>.
[66] T. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, ACM SIGCOMM Comp. Commun. Rev. 40 (1) (2010) 92–99.
[67] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC'10, November 2010, pp. 267–280.
[68] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, August 2012, pp. 7–12.
[69] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn, ACM SIGCOMM Comp. Commun. Rev. (2013) 99–110.
[70] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, G. Xie, The flowadapter: enable flexible multi-table processing on legacy hardware, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13, August 2013, pp. 85–90.
[71] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, Enabling fast failure recovery in openflow networks, in: Proceedings of 8th International Workshop on the Design of Reliable Communication Networks, DRCN'11, October 2011, pp. 164–171.
[72] D. Staessens, S. Sharma, D. Colle, M. Pickavet, P. Demeester, Software defined networking: Meeting carrier grade requirements,

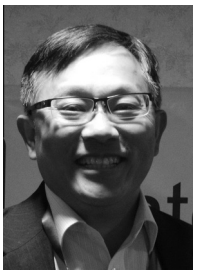
in: Proceedings of the 18th IEEE Workshop on Local & Metropolitan Area Networks, LANMAN'11, October 2011, pp. 1–6.
[73] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, Openflow: meeting carrier-grade recovery requirements, Comp. Commun. 36 (6) (2012) 656–665.
[74] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, P. Castoldi, Openflow-based segment protection in ethernet networks, in: IEEE/OSA Journal of Optical Communications and Networking Covers Advances in the State-of-the-Art of Optical Communications and Networks, vol. 5(9), September 2013, pp. 1066–1075.
[75] V. Sharma, Framework for Multi-Protocol Label Switching (MPLS)-Based Recovery, RFC 3469, Tech. Rep., February 2003.
[76] J.-P. Vasseur, M. Pickavet, P. Demeester, Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS, Elsevier, 2004.
[77] D. Katz, D. Ward, Bidirectional Forwarding Detection (bfd), RFC 5880, Tech. Rep., June 2010.
[78] M. Desai, T. Nandagopal, Coping with link failures in centralized control plane architectures, in: Proceedings of 2010 Second International Conference on Communication Systems and Networks, COMSNETS'10, January 2010, pp. 1–10.
[79] N. Budhiraja, K. Marzullo, F.B. Schneider, S. Toueg, The Primary-Backup Approach, Distributed Systems, vol. 2, second ed., 1993, pp. 199–216.
[80] P. Fonseca, R. Bennesby, E. Mota, A. Passito, A replication component for resilient openflow-based networking, in: Proceedings of Network Operations and Management Symposium, NOMS'12, April 2012, pp. 933–939.
[81] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-aware controller placement for software-defined networks, in: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), May 2013, pp. 672–675.
[82] Os3e <https://www.internet2.edu/news/detail/4865/>.
[83] N. Spring, R. Mahajan, D. Wetherall, Measuring isp topologies with rocketfuel, IEEE/ACM Trans. Network. 12 (2004) 2–16.
[84] Huawei technologies sdn showcase at sdn and openflow world congress 2013 <http://www.eantc.de/fileadmin/eantc/downloads/events/2011-2015/SDNOF2013/EANTC-Huawei_SDN_Showcase-White_Paper_Final_Secure.pdf>.
[85] M. Luo, et al., Dsox: Tech Report, Technical Report, Huawei Shannon Lab, May 2013.
[86] M. Reitblatt, N. Foster, J. Rexford, D. Walker, Consistent updates for software-defined networks: Change you can believe in!, in: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, HOTNETS-X, November 2011, p. 7.
[87] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, Abstractions for network update, in: Proceedings of the ACM SIGCOMM 2012, August 2012, pp. 323–334.
[88] R. McGeer, A safe, efficient update protocol for openflow networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, August 2012, pp. 61–66.
[89] N.P. Katta, J. Rexford, D. Walker, Incremental consistent updates, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13, August 2013, pp. 49–54.
[90] T. Mizrahi, Y. Moses, Time-based updates in software defined networks, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'3, August 2013, pp. 163–164.
[91] M. Tal, M. Yoram, Time-based Updates in Openflow: A Proposed Extension to the Openflow Protocol, Israel Institute of Technology, Technical Report, CCIT Report, vol. 835, July 2013.
[92] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte, Real time network policy checking using header space analysis, in: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, NSDI'13, April 2013, pp. 99–112.
[93] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: static checking for networks, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, April 2012, p. 9.
[94] A. Tootoonchian, M. Ghobadi, Y. Ganjali, Opentm: traffic matrix estimator for openflow networks, in: Proceedings of the 11th International Conference on Passive and Active Measurement, PAM'10, April 2010, pp. 201–210.
[95] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, Flowsense: monitoring network utilization with zero measurement cost, in: Proceedings of the 14th International Conference on Passive and Active Measurement, PAM'13, March 2013, pp. 31–41.
[96] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with opensketch, in: Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI'13, vol. 13, April 2013, pp. 29–42.
[97] J. Suh, T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for sdn," IBM Research Report, January 2014.
[98] A.C. Myers, Jflow: practical mostly-static information flow control, in: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'99, January 1999, pp. 228–241.
[99] N. Bandi, D. Agrawal, A. El Abbadi, Fast algorithms for heavy distinct hitters using associative memories, in: Proceedings of 27th International Conference on Distributed Computing Systems, ICDCS'07, June 2007, p. 6.
[100] Y. Zhang, S. Singh, S. Sen, N. Duffield, C. Lund, Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications, in: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, October 2004, pp. 101–114.
[101] R. Schweller, A. Gupta, E. Parsons, Y. Chen, Reversible sketches for efficient and accurate change detection over network data streams, in: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, IMC'04, October 2004, pp. 207–212.
[102] A. Kumar, M. Sung, J.J. Xu, J. Wang, Data streaming algorithms for efficient and accurate estimation of flow size distribution, ACM SIGMETRICS Perform. Eval. Rev. 32 (1) (2004) 177–188.
[103] G. Huang, A. Lall, C.-N. Chuah, J. Xu, Uncovering global icebergs in distributed monitors, J. Netw. Syst. Manage. 19 (1) (2011) 84–110.
[104] J. Sanjuàs-Cuxart, P. Barlet-Ros, N. Duffield, R.R. Kompella, Sketching the delay: tracking temporally uncorrelated flow-level latencies, in: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, IMC'11, November 2011, pp. 483–498.
[105] L. Jose, M. Yu, J. Rexford, Online measurement of large traffic aggregates on commodity switches, in: Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE'11, March 2011, p. 13.
[106] M. Moshref, M. Yu, R. Govindan, Resource/accuracy tradeoffs in software-defined measurement, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, August 2013, pp. 73–78.
[107] A. Khurshid, W. Zhou, M. Caesar, P. Godfrey, Veriflow: verifying network-wide invariants in real time, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, vol. 42, August 2012, pp. 49–54.
[108] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G.M. Parulkar, Can the production network be the testbed?, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, vol. 10, October 2010, pp. 1–14.
[109] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann, Ofrewind: enabling record and replay troubleshooting for networks, in: Proceedings of Usenix Annual Technical Conference, Usenix ATC'11, June 2011.
[110] E. Al-Shaer, S. Al-Haj, Flowchecker: configuration analysis and verification of federated openflow infrastructures, in: Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, SafeConfig'10, October 2010, pp. 37–44.
[111] E. Al-Shaer, W. Marrero, A. El-Atawy, K. ElBadawi, Network configuration in a box: towards end-to-end verification of network reachability and security, in: 17th IEEE International Conference on Network Protocols, 2009, ICNP 2009, IEEE, 2009, pp. 123–132.
[112] N. Foster, R. Harrison, M.J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, Frenetic: a network programming language, ACM SIGPLAN Not. 46 (9) (2011) 279–291.
[113] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, A nice way to test openflow applications, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, April 2012.
[114] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, N. McKeown, Where is the debugger for my software-defined network?, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, August 2012, pp. 55–60.
[115] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: experience with a globally-deployed software defined wan, in: Proceedings of the ACM SIGCOMM Conference, SIGCOMM'13, August 2013, pp. 3–14.

[116] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven wan, in: Proceedings of the ACM SIGCOMM 2013, August 2013, pp. 15–26.
[117] ns-2 <http://www.isi.edu/nsnam/ns/>.
[118] J. Zhang, K. Xi, M. Luo, H.J. Chao, Load balancing for multiple traffic matrices using sdn hybrid routing, in: Proceedings of IEEE 15th International Conference on High Performance Switching and Routing, Vancouver, July 1–4, 2014.
[119] J. Zhang, K. Xi, M. Luo, Dynamic hybrid routing: achieve load balancing for changing traffic demands, in: Proceedings of the IEEE/ACM IWQoS 2014, Hong Kong, May 2014.
[120] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, R. Johari, Plug-n-serve: Load-balancing web traffic using openflow, Demo at ACM SIGCOMM, August 2009.
[121] N. Handigol, S. Seetharaman, M. Flajslik, A. Gember, N. McKeown, G. Parulkar, A. Akella, N. Feamster, R. Clark, A. Krishnamurthy, et al., Aster*x: load-balancing web traffic over wide-area networks, 2009.
[122] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, Flowvisor: A Network Virtualization Layer, OpenFlow Switch Consortium, Technical Report, October 2009.
[123] Flowvisor <http://onlab.us/flowvisor.html>.

Ian F. Akyildiz received the B.S., M.S., and Ph.D. degrees in Computer Engineering from the University of ErlangenNrnberg, Germany, in 1978, 1981 and 1984, respectively. Currently, he is the Ken Byers Chair Professor in Telecommunications with the School of Electrical and Computer Engineering, Georgia Institute of Technology (Georgia Tech), Atlanta, GA USA; the Director of the Broadband Wireless Networking (BWN) Laboratory and the Chair of the Telecommunication Group at Georgia Tech. Since 2013, he is a FiDiPro Professor (Finland Distinguished Professor Program (FiDiPro) supported by the Academy of Finland) in the Department of Electronics and Communications Engineering, at Tampere University of Technology, Finland, and the founding director of NCC (Nano Communications Center). Since 2008, he is also an honorary professor with the School of Electrical Engineering at Universitat Politcnica de Catalunya (UPC) in Barcelona, Catalunya, Spain, and the founding director of N3Cat (NaNoNetworking Center in Catalunya). Since 2011, he is a Consulting Chair Professor at the Department of Information Technology, King Abdulaziz University (KAU) in Jeddah, Saudi Arabia. He is the Editor-in-Chief of Computer Networks (Elsevier) Journal, and the founding Editor-in-Chief of the Ad Hoc Networks (Elsevier) Journal, the Physical Communication (Elsevier) Journal and the Nano Communication Networks (Elsevier) Journal. He is an IEEE Fellow (1996) and an ACM Fellow (1997). He received numerous awards from IEEE and ACM. His current research interests are in nanonetworks, Terahertz Band communication networks, Long Term Evolution Advanced (LTE-A) networks, cognitive radio networks and wireless sensor networks.

Ahyoung Lee received the M.S., and Ph.D. degrees in Computer Science and Engineering from the University of Colorado, Denver, CO USA in 2006 and 2011, respectively, and B.S. degree in Information and Computer Engineering from the Hansung University in 2001, Seoul, Korea. She was a Senior Researcher in the Communication Policy Research Center at the Yonsei University, Seoul, Korea in 2012. Currently, she is a Postdoctoral Fellow at the Georgia Institute of Technology, in the Broadband Wireless Networking Laboratory (BWN Lab) under the supervision of Prof. Dr. Ian F. Akyildiz with a research project focused on Software Defined Networking (SDN). Her main research interests include adaptive routing schemes for large-scale network resources, analytical models and network performance evaluations in Ad Hoc Wireless Networks, Sensor Networks and Mobile Wireless Networks; future internet architecture for wireless/mobile cloud networking; securing wireless applications and networks.

Pu Wang received the B.E. degree in Electrical Engineering from Beijing Institute of Technology, China, in 2003, and the M.E. degree in Electrical and Computer Engineering from Memorial University of Newfoundland, Canada, in 2008. He received the Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta, GA USA, in August 2013, under the guidance of Prof. Dr. Ian F. Akyildiz. Currently, he is an Assistant Professor with the Department of Electrical Engineering and Computer Science at the Wichita State University. He received the Broadband Wireless Networking Laboratory (BWN Lab) Researcher of the Year Award at the Georgia Institute of Technology in 2012. He received the TPC top ranked paper award of IEEE DySPAN 2011. He was also named Fellow of the School of Graduate Studies, Memorial University of Newfoundland in 2008. He is a member of the IEEE. His current research interests are wireless sensor networks, cognitive radio networks, software defined networking, Internet of multimedia things, nanonetworks, and wireless communications in challenged environment.



Min Luo received the Ph.D. degree in Electrical Engineering from Georgia Institute of Technology, Atlanta, GA USA, in 1992. He also held the B.S., and M.S. degrees in 1982 and 1987, respectively in Computer Science. Currently, he is the Head and Chief Architect of the Advanced Networking at Huawei's Shannon (IT) Lab, leading the research and development in Software Defined Networking (SDN) and other future networking initiatives. He served as Chief/Executive Architect for IBM SWG's Strategy and Technology, Global Business Solution CenterGCG, Industry Solutions, and Center of Excellence for Enterprise Architecture and SOA for more than 11 years. He also worked as Senior Operations Research Analyst, Senior Manager and Director of Transportation Network Planning and Technologies for two Fortune 500 companies for 7 Years. He's certified and awarded as the Distinguished Lead/Chief Architect from Open Group in 2008. He is an established expert in the field of next negation software defined networking (SDN), enterprise architecture and information systems, whole life cycle software application and product development, business intelligence, and business process optimization. He is also a pioneer and one of the recognized leading experts and educators in Service-oriented architecture (SOA), Model/business-driven architecture and development (MDA-D), and component/object-oriented technologies. He coauthored 2 books, including the pioneering Patterns: Service Oriented Architecture and Web Services in 2004, and published over 20 research papers. As a senior member of IEEE, he has been serving on the organizing committee for IEEEs ICWS, SCC and CC (Cloud Computing) Conferences, chaired sessions, presented several tutorials on SOA and Enterprise Architecture and their best practices and gave lectures at the Service University. He has served as adjunct professors in several USA and Chinese universities since 1996.

Wu Chou received the Ph.D. degree with four advanced degrees in Science and Engineering from the Stanford University, CA USA in 1990. Currently, he is VP, Chief IT Scientist, and Global Head of Huawei Shannon (IT) Lab, USA. He is an IEEE Fellow, a renowned expert in the field of IT, computing, networking, Internet/Web, Big Data, SDN (software-defined-network), communication, signal processing, speech and natural language processing, machine learning, unified communication, smart systems and endpoints. He has over 20+ years of professional career in leading R&D organizations. Before joining Huawei, he was Director of R&D at Avaya. He joined AT&T Bell Labs after obtaining his Ph.D. degree and continued his professional career from AT&T Bell Labs to Lucent Bell Labs and Avaya Labs before joining Huawei. In his role at Huawei, he leads the global Huawei Shannon (IT) Lab in its

research and innovation in the fast moving IT area. He has extensive experience in cutting-edge technology research, incubating ground breaking products, visionary technical leadership, and agile execution in research and product development. He published over 150 journal and conference papers, holds 32 USA and international patents with many additional patent applications pending. He received Bell Laboratories Presidents Gold Award for his achievement in 1997, Avaya Leadership Award in 2005, and the outstanding standard and patent contribution award in 2008 and 2009. He is a well known figure in standard bodies and professional societies. He served as an editor for multiple standards at W3C, ECMA, ISO, ETSI, etc. He was an editor of IEEE Transactions on Services Computing (TSC), IEEE TSC Special Issue on Cloud Computing, IEEE Transaction on Audio and Language Processing, and Journal of Web Services Research.