Research Challenges for Traffic Engineering in Software Defined Networks

Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou

Abstract

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 costs, and promote innovation and evolution. While traffic engineering techniques have been widely exploited for ATM and IP/MPLS networks for performance optimization in the past, the promising SDN networks require novel traffic engineering solutions that can exploit the global network view, network status, and flow patterns/characteristics in order to achieve better traffic control and management. This article discusses the state-of-the-art in traffic engineering for SDN with attention to four cores including flow management, fault tolerance, topology update, and traffic analysis. Challenging issues for SDN traffic engineering solutions are discussed in detail.

The recently emerged software defined networking (SDN) [1] architecture separates between the network control plane and the data plane, which provides user applications with a centralized view of the distributed network states. A logical view of the SDN architecture is depicted in Fig. 1. At the control plane layer, there is one SDN controller; however, there may be many controllers in a large-scale or wide area network. The network intelligence and states are handled by controller(s), where the controller(s) can globally regulate the network states via network policies in either a centralized (*one controller*) or distributed manner (*many controllers*). A set of application programming interfaces called north-bound open application programming interfaces (APIs) are supported to communicate between the application layer and the control plane layer in order to enable network services. The data plane layer in SDN employs programmable OpenFlow (OF) switches that communicate with its SDN controller via south-bound open interfaces (e.g., OF protocol). The OF protocol [1] allows the logically centralized controller to dynamically modify the forwarding table of routers or switches, regardless of the underlying switching technologies. The SDN paradigm offers a unified and global view of complicated networks, and thus provides a powerful control environment for network management of traffic flows. So far in the literature, the majority of research was devoted mainly to the development of SDN architectures, with less effort on the development of traffic engineering (TE) tools for SDN.

Traffic engineering is an important subject for network performance optimization by dynamically analyzing, predicting, and regulating the behavior of the transmitted data. TE has been widely exploited in asynchronous transfer mode (ATM) and IP/multiprotocol label switching (MPLS) networks over the last two and half decades. However, those networking paradigms and their corresponding TE solutions are not directly useful in the next generation networking paradigms such as SDN. This is mainly due to two reasons:

- Today's Internet applications require the underlying network architecture to behave in real time (or near real time) and to scale up to a large amount of traffic. The network architecture should be capable of classifying a variety of traffic types for different applications, and to provide an appropriate and specific service for each traffic type in a very short time period such as within milliseconds.
- Highly efficient network management is desirable to significantly improve resource utilization for optimal system performance when it encounters the rapid growth in cloud computing and the demand for massive-scale data centers. However, all the existing TE technologies rely on closed and inflexible architectural design, where the control and data planes are tightly coupled and integrated. Such inflexible and close architectures prevent the existing TE technologies providing truly differentiated services to adapt to increasingly growing, uneven, and highly variable traffic patterns. On the contrary, the unique features of SDN, including visibility, programmability, openness, and virtualizability, pave the way for the development of new TE techniques that are inherently flexible, adaptive, and customizable.

In this article, we survey the state of the art in TE for SDN from the perspective of four thrusts: flow management, fault tolerance, topology update, and traffic analysis. We also highlight open challenging research issues and review recent progress in TE for SDNs.

Scalability and Availability: Flow Management

In SDN, when an OF switch receives a new (*unknown*) flow that does not match any rule in its flow entry, the first packet of this flow is forwarded to its controller. Accordingly, the controller then calculates a forwarding path and installs a new forwarding rule for the switches along that path. However, installing this forwarding rule may take time and yield delay spikes. If a high number of new flows are aggregated at the

Ian F. Akyildiz and Ahyoung Lee are with Georgia Institute of Technology.

Pu Wang is with Wichita State University.

Min Luo and Wu Chou are with Huawei Technologies Co. Ltd.

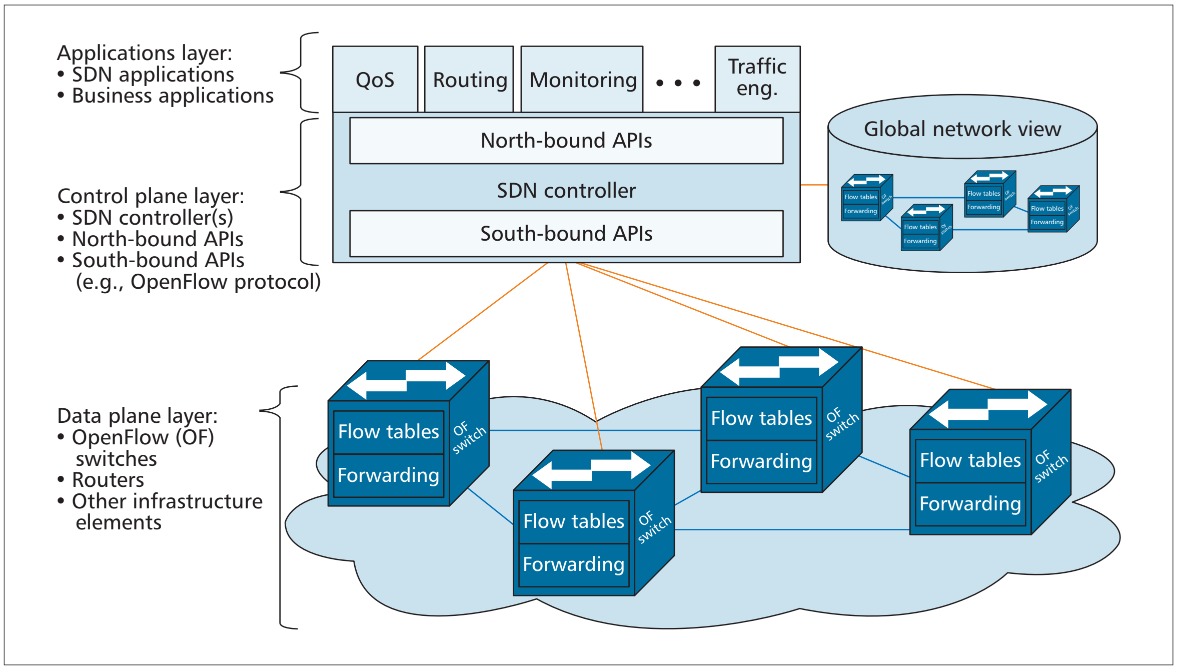

Figure 1. Overview of SDN architecture.

edge switches, significant overhead may arise at both the control and data planes. Thus, we present solutions that avoid this bottleneck in SDNs by considering the trade-offs between latency and load balancing at the data plane and the control plane.

Load Balancing for the Data Plane

Hash-Based ECMP Flow Forwarding: The hash-based Equal-Cost Multi-Path (ECMP) [2] is a load balancing scheme to distribute flows across available paths using flow hashing methods. However, a main constraint of ECMP is that two or more large, long-lived flows can collide on their hash and thus share the same output port, thereby creating a bottleneck in the network. This static mapping of flows to paths is not concerned with either current network utilization or flow size, thus resulting in collisions that can overwhelm the switch buffers and degrade the overall network utilization. To avoid the limitations of ECMP, a significant amount of large (*elephant*) flows can be detected at the edge switches [3] or even at the end hosts [4], and then the central controller could calculate the appropriate paths for them, while small (mice) flows are forwarded by using the ECMP routing at the switches. However, such a solution can induce high bandwidth and processing overhead at the switches or hosts.

Wildcard Rule Flow Forwarding: OF switches use flow-match wildcards to aggregate traffic flows [2]. OF is a great concept that simplifies network and traffic management by enabling flow-level control over switches and providing a global network view. However, the central control management and the global view over all flows require the controller to set up all flows for the critical path in the entire network, which is not sufficiently scalable and results in both bottleneck and latency. To reduce the number of interactions between the controller and the switches, the SDN TE approaches implement wildcard OF rules at the switches, and the switches can make local routing decisions that handle mice flows to avoid involving the controller, while the controller maintains the control over only targeted elephant flows, especially for quality of service (QoS)-significant flows. Another approach [2] uses the authority switches that can handle all data packets without involving the controller to reduce the control overhead at the control plane.

Load Balancing for the Control Plane

Whenever a new flow is established in the network, the OF switch has to forward the first packet of the flow to the controller for deciding a suitable forwarding path. This feature of SDN may cause the network controller to become a potential performance bottleneck. A single controller cannot work efficiently and scale up with the increased number of network elements and the growing number of traffic flows. Toward this end, controller load balancing schemes have to be adopted, which are summarized as follows.

Distributed Controller Deployment:

Logically Centralized and Physically Distributed Controller: The logically centralized control plane aims to keep the benefit of network control centralization by using a set of physically distributed controllers. For this, there are mainly two approaches. The first one, HyperFlow [2], localizes the decision making to individual controllers and uses a publish/subscribe messaging paradigm with a distributed file system to provide the same consistent network-wide view to all of its controllers. However, this scheme requires additional maintenance and subscription management that may increase the control overhead. In contrast, the second approach, DIFANE [2], distributes the controller's rules to a subset of the authority switches, which handle the traffic flow forwarding decisions in the data plane. This scheme may have high resource consumption such as CPU or the ternary content addressable memory (TCAM) [2] at switches.

Physically Distributed Controller: By this deployment scheme, a large-scale OF network is partitioned into small networks, each of which is managed by a local controller. For example, in the Onix system [2], each local controller has the network information base (NIB) data structure to share the copy of the network state with each other. The NIB includes a graph of all network entities within a local network topology. BalanceFlow [2] is a more flexible controller load balancing architecture for a wide-area OF network. The super controller is in charge of balancing the load of all controllers, while all controllers maintain their own flow requests information and publish this information periodically through a cross-controller communication system to support load balancing.

Multi-thread approach [2]	OF version	Number of threads used in CPU cores	Maximum throughput	Average delay
Maestro	v1.0	7 (8 cores from 2 × Quad-Core AMD Opteron 2393 processors)	0.63 million flow requests per second (rps)	76 ms
Beacon	v1.0	12 (16 cores from 2 × Intel Xeon E5-2670 processors)	12.8 million flow rps	0.02 ms
NOX-MT	v1.0	8 (8 cores from 2 GHz processor)	1.6 million flow rps	2 ms
SOX	v1.3+	4 (4 cores from 2.4 GHz processor)	0.9 million flow rps per server, 3.4+ million flow rps with 4 servers in the cluster while hitting the I/O limit	< 0.5 ms (end-to-end with 2 tandem switches)

Table 1. Quantitative overview of multi-thread controllers.

Hierarchical Controller: This is a hierarchically distributed control plane with different levels. Kandoo [2] is an example of a hierarchical controller deployment. A network controlled by Kandoo has multiple local controllers and a logically centralized root controller. If the root controller needs to set up flow entries on switches of a local controller, it forwards the requests to the respective local controller. However, in this hierarchical controller architecture, the local controller needs to have a global network view of their applications.

Hybrid Controller: It is a logically centralized control plane, but physically distributed clusters of controllers. SOX [5] and the distributed SOX (DSOX) are designed with a centralized controller clusters while many controllers could be concurrently running in equal mode and the cluster shares a common NIB. This architecture enables automatic failover and load balancing, while the number of controller instances is created dynamically according to the changing traffic demands.

Multi-Thread Controllers: To improve the request processing throughput, multi-thread multi-core SDN controllers have been developed, where the parallelism architecture of servers is exploited to provide high throughput with scalability at the controller. By default, almost all production controllers nowadays are multi-threaded in order to provide adequate performance. Several multi-thread controllers are proposed, and the performance evaluation results depend on their tested conditions as shown in Table 1.

Generalized Controllers: To improve the flexibility, reliability, and advanced networking capabilities, the subsequent standard releases of OpenFlow after OF v1.0 [1] have gradually introduced many core protocol level enhancements (e.g., multi-flow tables and multi-controllers) in addition to other critical new networking features including IPv6, MPLS, flow metering, and so on. However, these desired new capabilities come at a cost, because the increased protocol complexity significantly affects the design and implementation of both the controllers and switches. Moreover, in the foreseeable future, SDN/OF-based networking technologies will coexist and probably interoperate with existing IP-based ones. This interoperating issue also occurs with the SDN/OF-based switches and controllers, as different OF standard versions (e.g., OF v1.0 and OF v1.2) are not backward compatible. To cope with such a problem, SOX [5] initiated the approach of a generalized SDN controller to control the SDN/OF-based data networking. It supports interoperation of both OF v1.0 and v1.2+ switches in addition to internetworking with existing legacy data networks. Moreover, in order to support the structured controller evolution, it promotes and adopts the model-driven architecture based on best software engineering practice to improve the extensibility, modularity, usability, interoperability, consistency, and manageability of SDN. SOX is multi-threaded and can be deployed in a clustered environment in equal-equal mode, in which the number of threads in SOX is dynamically adjusted and fluctuated with the level of data traffic (packet-in rates) to the controller.

Multiple Flow Tables

The OF v1.0-based switches [1] have a single match table model typically built on TCAM. However, the single table for implementing flow rules can create a huge rule set, resulting in a serious limitation on the number of flow entries and the inability to support large-scale deployments, since TCAM space is a scarce and expensive resource. It is inefficient to store many attributes in a single table with tremendous redundancy, which can degrade searching and matching speed as well. To make flow management more flexible and efficient, OF v1.1(+) [1] introduces the mechanism of multiple flow tables. As shown in Fig. 2, an OF switch can have one or more flow tables in the switch for pipelined processing. Decomposing the single flow table into multiple more efficient and normalized sets of tables can significantly improve TCAM resource utilization and speed up the matching process.

Research Challenges

Dynamic Load-Balancing Scheme for the Data Plane: To fully utilize the flexible control and global view promised by SDN, SDN TE demands a dynamic load balancing mechanism that is adaptive to time-varying network states and adjustable based on fine-grained traffic characteristics such as traffic burstiness and inter-arrival times.

Dynamic Load Balancing Scheme for the Control Plane: To avoid the bottleneck at the single centralized controller in a large-scale SDN network, TE should consider control plane load balancing solutions to find the optimal number, locations, workload distribution, and control message forwarding paths of SDN controllers. The traffic balancing solutions have to facilitate efficient and accurate acquisition of the traffic statistics in SDN.

Adaptive Multi-Flow Table Schemes: As the number of flows managed by a switch is limited by the size of its flow tables, flexible and adaptive flow table methods should be developed so that the new flows which overwhelmed the limited space of TCAM(s) can be moved to the attached memory space of lower-cost SRAM or DRAM. These methods should be connected with an efficient traffic scheduling method for different QoS flows.

Reliability: Fault Tolerance

To ensure network reliability, SDN should be able to perform failure recovery transparently and gracefully when failures occur in the network infrastructure. Although a switch could

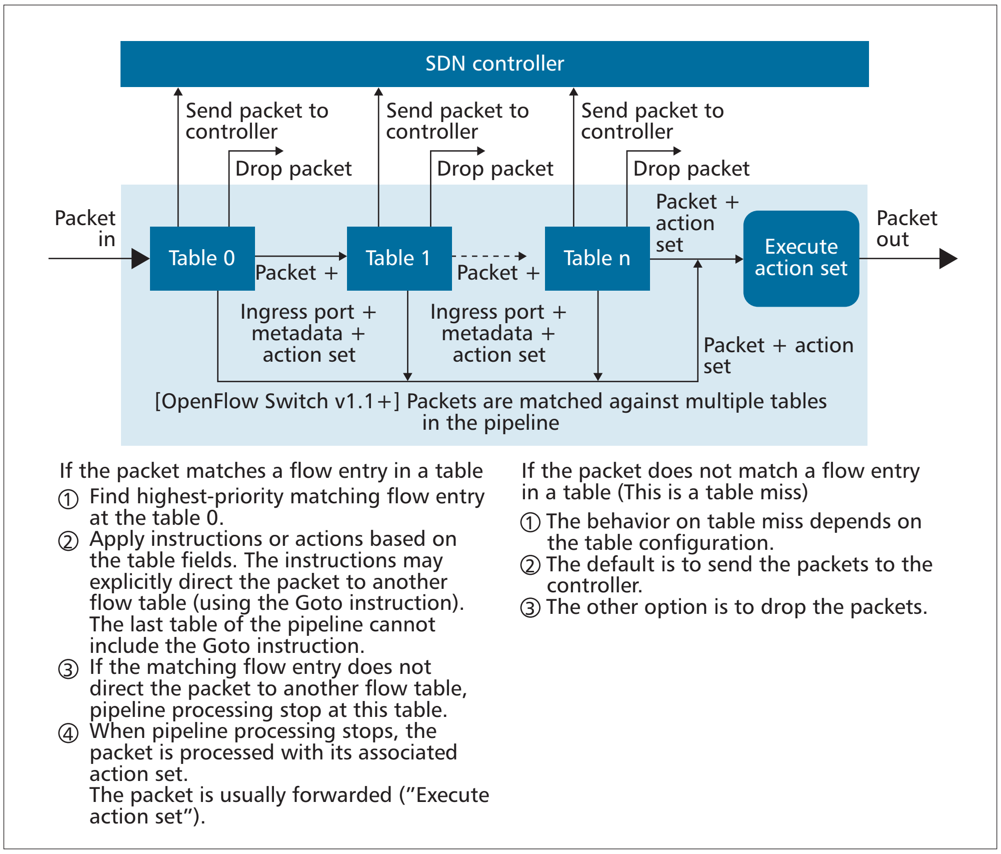
SDN controller
Send packet to controller
Send packet to controller
Send packet to controller
Drop packet
Drop packet
Drop packet
Packet in
Table 0
Table 1
Table n
Packet +
Packet +
Packet + action set
Execute action set
Packet out
Ingress port + metadata + action set
Ingress port + metadata + action set
Packet + action set
[OpenFlow Switch v1.1+] Packets are matched against multiple tables in the pipeline
If the packet matches a flow entry in a table
① Find highest-priority matching flow entry at the table 0.
② Apply instructions or actions based on the table fields. The instructions may explicitly direct the packet to another flow table (using the Goto instruction). The last table of the pipeline cannot include the Goto instruction.
③ If the matching flow entry does not direct the packet to another flow table, pipeline processing stop at this table.
④ When pipeline processing stops, the packet is processed with its associated action set. The packet is usually forwarded ("Execute action set").
If the packet does not match a flow entry in a table (This is a table miss)
① The behavior on table miss depends on the table configuration.
② The default is to send the packets to the controller.
③ The other option is to drop the packets.

Figure 2. Packet flow over multiple flow table pipelines under OF v1.1+.

identify the failed link, it has neither the intelligence nor the global knowledge to create a new route. It has to depend on updates from the controller to establish an alternate route. Moreover, when the failed node is recovered and goes back to work, it will still be the duty of the controller to re-establish the optimal routes and the network topology for the ongoing traffic. Therefore, we investigate current research efforts and suggestions on realizing fast failure recovery in SDN networks.

Fault Tolerance for the Data Plane

Failure Recovery Mechanisms: There are primary two types of failure recovery mechanisms: restoration and protection for the network element and link failures [6]. Restoration is a reactive strategy, while protection is a proactive strategy.

Restoration: The recovery paths can be either pre-planned or dynamically allocated, but resources are not reserved until failure occurs. Additional signaling is required to establish the restoration path when a failure occurs.

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.

Compared to the restoration scheme, the protection scheme [7] can enable faster recovery without the involvement of the network controller when failures are detected. Moreover, the required bandwidth and latency during failures can be considerably reduced because no interactions are required between switches and the controller. Therefore, for large-scale SDN systems, path protection solutions are more favorable in order to achieve fast failure recovery.

Other Considerations for Fast Failure Recovery: The delay in failure recovery can also be caused by the OF protocol. According to the OF specification [1], even if the new flow entries are installed at the affected switch, the switch does not remove the entries using the failed link until the timeout occurs. The timeout is associated with timers (i.e., hard timer and soft timer), and it is normally in the range of several seconds. Thus, the path failures are not actually recovered until one of the timers expires. To solve these problems, the OF-based segment protection scheme [7] employs the flow entry priority and auto-reject mechanism to achieve fast switchover between the working path and protection path. Upon detecting failures, the auto-reject mechanism removes all affected flow entries using the failed links immediately without waiting for either the soft or hard timeout. Reference [8] proposed a scheme that allows switches to exchange simple link failure messages (LFMs) in such a way that the relevant switches can be aware of a link failure without involving the controller, thus leading to a much shorter time than it would take for the controller to identify a link failure and send out the topology update to the relevant switches. The performance of this scheme depends on the number of switches and also on the total number of flow table entries in a switch.

Fault Tolerance for the Control Plane

SDN is a logically centralized architecture, which depends on the controller to update policies and take actions when new flows are introduced in the network. Therefore, the reliability of the control plane is critically important. Without resolving a single point of failure in the control plane, the performance of the entire network may be significantly degraded. The most fundamental mechanism to recover control plane failures in a centralized network is the "primary-backup replication" method, where the backup controllers take over the network control and operation when the primary controller fails.

Primary and Backup Controller Coordination: The OF protocol has limited ability to configure one or more backup controllers. However, the OF protocol does not provide any coordination mechanism between the primary controller and the backups. Thus, coordination protocols are desired, which not only are able to perform the coordination between controllers to keep the backup controller consistent with the primary one, but also return the network to a safe state with minimal overhead imposed on switches and hosts. To support the primary backup mechanism, CPRecovery [9] employs the replication process between the switch component running on the primary controller and the secondary controller by using probe messages sent from switches. If the controller does not send a reply for the probe within the waiting time, the switch assumes that the controller is down. The switch searches for the next secondary controller acting as a backup in its list, and the secondary controller becomes a primary controller after it receives a connection request from the switch.

Backup Controller Deployment: Properly placing backup controllers in SDN can help maximize network reliability. The impact of the number of the controllers on network reliability needs to be determined, and the trade-offs between reliability and latency should be considered. According to the analysis results [10], k controllers reduce latency to $1/k$ of the original single-controller latency, and the analysis results indicate that deploying more than three controllers cannot further reduce the latency. The controller placement problem is further studied in [11]. It is suggested that the best controller placement is to use one controller that yields the optimal reliability metric while optimizing the average latency.

Research Challenges

Cost-Efficient and Fast Failure Recovery for the Data Plane: Fast failure recovery mechanisms should be implemented so that it can be achieved with low communication overhead and less/no interference to the controller, and requires minimum intelligence at the switches.

Primary-Backup Replication with Traffic Adaptivity for the Control Plane: To achieve high reliability and optimal performance of SDN controller(s), it is important to find an optimal number of controllers and their best locations for the primary controller as well as the backup controller(s) in the context of an optimal trade-off between reliability and latencies for time-varying traffic patterns, including traffic volume trends in the entire network and so on.

Consistency: Topology Update

This section is focused on planned changes such as network policy rules changes. General update operations are implemented: each packet/flow is identified when updating the network from the old policy to the new policy over multiple switches, and then is guaranteed to be managed by either the old policy or the new policy, but not by the combination of the two [12]. There are primarily two types of consistency: per-packet consistency, in which each packet flowing through the network is processed according to a single network configuration, and per-flow consistency, in which all packets in the same flow are managed by the same version of the policy; thus, per-flow abstraction preserves all path properties.

Duplicate Table Entries in Switches

To implement a consistent per-packet and per-flow update, some simple generic ideas are proposed in [12]. The key common operation of per-packet/-flow consistent updates is that the switches process a packet following either the old or new policy until the controller deletes the old configuration rules from all switches. The major problem of such a duplicate policy scheme is that it requires holding both old and new sets of rules on the network switches. Thus, the efficiency of these algorithms depend on explicit information of how long the switches need to hold the old rules due to the limited memory (e.g., the limited TCAM space of switches). To address this problem, more efficient update algorithms are required for implementing consistent updates from the old rules to the new ones with high flow initiation rate between the controller and the switches, and for an optimal trade-off between update time and rule-space overheads.

Time-Based Configuration

Reference [13] presents a method to allow coordinated SDN network updates in multiple switches based on a time-based sequence of different update times. However, the controller must first wait for an acknowledgment from the switch to complete the update and then send the new policy to other switches until the network is completely updated in the OF network. To solve this problem Net-Plumber [14] was proposed to configure the forwarding table with significantly fast update time. Rather than updating all the switches simultaneously, it incrementally updates only the portions of the switches affected by the changing rules in the network using a plumbing graph, which caches all possible paths of flows over the network to immediately update the reachable switches of a path for the flow, which is filtered by the OF rule (e.g., match, action).

Research Challenges

A Single Controller in a Large-Scale SDN Network: SDN may experience control packet loss because network congestion can induce memory/buffer overflows in OF switches. The loss of a control packet degrades the consistency of the network policies. How a single controller can efficiently update the network information with high consistency in the presence of control packet loss is still an open problem.

Multiple Controllers in Multi-Domain SDN Networks: How to consistently update the shared network information in the entire network with the optimal trade-offs between the low inter-synchronization overhead and the real-time update needs to be addressed.

Accuracy: Traffic Analysis

Monitoring Framework

Network monitoring is of crucial importance for network management. Management applications need accurate and timely statistics on network resources at different aggregation levels (e.g., flow, packet, and port) [15]. SDN networks must continuously monitor the performance metrics, such as link utilization, to immediately adapt forwarding rules to the changes in workload. However, existing monitoring solutions either require special instrumentation of the network or impose significant measurement overhead (e.g., NetFlow, sFlow, and JFlow) [2]. These monitoring approaches may not be efficient for application in the SDN architecture, such as large-scale data center networks, because of the high overhead caused by the collection of the statistics from the whole network at the central controller. Thus, the current monitoring solutions in SDN seek more efficient monitoring mechanisms in order to achieve both high accuracy and low overhead. The current solutions are classified into two categories. *Query-based monitoring* is based on the request/response paradigm that periodically or adaptively polls the switch on each active flow for collecting flow-level statistics, thus yielding high accuracy along with high overhead. In this case, most SDN TE solutions use the wildcard rules at the switches to only monitor aggregated flows, instead of individual flows, to extract significant traffic patterns with minimum monitoring overhead. *Push-based monitoring* is based on the publish/subscribe/distribute paradigm, where the server automatically pushes (delivers) information to clients without repeated requests from the clients. Thus, the number of client requests handled by a server can be reduced dramatically. In this scheme, a monitoring tool is implemented in a dedicated server separate from the network controller. Such a solution can yield accurate traffic monitoring with low latency, while considerably reducing the processing overhead at the controller for collecting flow statistics.

Checking Network Errors

SDN makes the network open to applications, allowing multiple applications and 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. The most common verification tools of SDN/OF are implemented as a proxy residing between the controller and switches for monitoring all communication and verifying network-wide invariant violations as each forwarding rule is inserted. To achieve real-time checking in SDN networks, VeriFlow [2] slices the network into a set of equivalent classes of packets based on the destination IP address, which can be verified within hundreds of microseconds as new rules are installed into the network. Also, the verification tool can allow OF administrators/users to manually verify the consistency of multiple controllers and switches across different OF federated infrastructure.

Debugging Programming Errors

Current networks provide a variety of interrelated services including routing, load balancing, traffic monitoring, access control, and so on. They are commonly defined at the low level of abstractions offered by the underlying hardware, thus often failing to provide even simple and basic support for modular programming. Therefore, network programs tend to be complicated, error-prone, and difficult to maintain.

To address this situation, NICE [2] was proposed as a tool that combines model checking and symbolic execution to efficiently discover violations of network-wide correctness properties due to bugs in the controller programs. ndb [2] is a debugging software tool for SDN programmers/operators to track down the root cause of a bug. Thus, ndb can detect bugs in any level of the SDN architecture and provide a finer-grained debugging environment than NICE.

Research Challenges

Traffic Analysis: How to efficiently handle big data in the context of user behavior, locality, and time-dependent statistics, especially from mobile applications, in SDN-TE needs to be addressed.

Traffic Monitoring: How to significantly reduce the network overhead when SDN controller(s) or monitoring devices collect the network statistics with high accuracy is a topic for further study.

Network Checking and Programming Debugging Methods: The verification and debugging methods should work together to address network security issues. Methods to quickly detect or prevent intrusions by using network verification or programming error checking approaches for SDN are still largely unexploited. (Note that we do not account for security mechanisms in this article because the topic is beyond the scope of TE. However, there are a plethora of open research problems regarding the security aspects in SDNs.)

Conclusion

SDN represents a new, flexible, and open architecture that allows the dynamic and timely regulation of the behavior of network switches in complex and large-scale computer networks. As SDN is accelerating the innovation and evolution of modern data networks, it requires a highly scalable and intelligent TE system. This article investigates the SDN TE solutions from the perspectives of flow management, load balancing, fault tolerance, topology update, and traffic analysis. The current state and research challenges of SDN TE are presented by addressing the key SDN performance metrics in terms of scalability, availability, reliability, consistency, and accuracy.

References

[1] "OpenFlow Switch Specification v1.0-v1.4"; www.opennetworking.org/sdn-resources/onf-specifications.

[2] I. F. Akyildiz *et al.*, "A Roadmap for Traffic Engineering in Software Defined Networks," *Computer Networks*, vol. 71, Oct. 2014, pp. 1–30.

[3] M. Al-Fares *et al.*, "Hedera: Dynamic Flow Scheduling for Data Center Networks," *Proc. Networked Systems Design and Implementation Symp.*, vol. 10, Apr. 2010, p. 19.

[4] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-Overhead Datacenter Traffic Management Using End-Host-Based Elephant Detection," *Proc. 30th IEEE INFOCOM '11*, Apr. 2011, pp. 1629–37.

[5] M. Luo *et al.*, "Sox: Generalized and Extensible Smart Network Openflow Controller (x)," *Proc. First SDN World Congress*, Damsdadt, Germany, Oct. 2012.

[6] J.-P. Vasseur, M. Pickavet, and P. Demeester, *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*, Morgan Kaufmann, 2004.

[7] A. Sgambelluri *et al.*, "Openflow-Based Segment Protection in Ethernet Networks," *IEEE/OSA J. Opt. Commun. Net*, vol. 5, no. 9, Sep. 2013, pp. 1066–75.

[8] M. Desai and T. Nandagopal, "Coping with Link Failures in Centralized Control Plane Architectures," *Proc. 2nd Int'l. Conf. Commun. Systems and Net.*, Jan. 2010, pp. 1–10.

[9] P. Fonseca *et al.*, "A Replication Component for Resilient Openflow-Based Networking," *Proc. IEEE NOMS '12)*, Apr. 2012, pp. 933–39.

[10] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," *Proc. 1st ACM Wksp. Hot Topics In Software Defined Networks*, Aug. 2012, pp. 7–12.

[11] Y. Hu et al., "Reliability-Aware Controller Placement for Software-Defined Networks," *Proc. IFIP/IEEE Int'l. Symp. Integrated Network Management*, May 2013, pp. 672–75.

[12] M. Reitblatt *et al.*, "Consistent Updates for Software-Defined Networks: Change You Can Believe In!" *Proc. 10th ACM Wksp. Hot Topics in Networks*, Nov. 2011, pp. 7.

[13] T. Mizrahi and Y. Moses, "Time-Based Updates in Software Defined Networks," *Proc. 2nd ACM SIGCOMM Wksp. Hot Topics in Software Defined Networking*, Aug. 2013, pp. 163–64.

[14] P. Kazemian *et al.*, "Real Time Network Policy Checking Using Header Space Analysis," *Proc. 10th USENIX Conf. Networked Systems Design and Implementation*, Apr. 2013, pp. 99–112.

[15] S. R. Chowdhury *et al.*, "Payless: A Low Cost Network Monitoring Framework for Software Defined Networks," *Proc. IEEE NOMS '14*, May 2014, pp. 1–9.

Biographies

IAN F. AKYILDIZ [F'96] received B.S., M.S., and Ph.D. degrees in computer engineering from the University of Erlangen Nürnberg, 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; 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 of the Finland Distinguished Professor Program (FiDiPro) supported by the Academy of Science) with the Department of Electronics and Communications Engineering, Tampere University of Technology, Finland, and the founding director of the Nano Communications Center (NCC). Since 2008, he is also an honorary professor with the School of Electrical Engineering at Universitat Politécnica de Catalunya (UPC), Spain, and the founding director of NaNoNetworking Center in Catalunya (N3Cat). Since 2011, he is a consulting chair professor at the Department of Information Technology, King Abdulaziz University (KAU), Jeddah, Saudi Arabia. He is Editor-in-Chief of the *Computer Networks Journal* (Elsevier), and the founding Editor-in-Chief of the *Ad Hoc Networks Journal* (Elsevier), the *Physical Communication Journal* (Elsevier), and the *Nano Communication Networks Journal* (Elsevier). He is an ACM Fellow (1997). He has received numerous awards from IEEE and ACM. His h-index is 88, and the total number of citations was above 69K according to Google Scholar as of March 2015. His current research interests are in software defined networks, nanonetworks, terahertz band, 5G cellular systems, and wireless sensor networks in challenged environments.

AHYOUNG LEE received her M.S., and Ph.D. degrees in computer science and engineering from the University of Colorado, Denver, in 2006 and 2011, respectively. Currently, she is a postdoctoral fellow at Georgia Tech in the BWN Lab under the supervision of Prof. Ian F. Akyildiz with a research project focused on 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; and securing wireless applications and networks.

Pu Wang [M] received his B.E. degree in electrical engineering from Beijing Institute of Technology, China, in 2003, and his M.E. degree in electrical and computer engineering from Memorial University of Newfoundland, Canada, in 2008. He received his Ph.D. degree in eectrical and computer engineering from Georgia Tech in August 2013, under the guidance of Prof. Ian F. Akyildiz. Currently, he is an assistant professor with the Department of Electrical Engineering and Computer Science at Wichita State University. He received the BWN Lab Researcher of the Year Award at Georgia Tech in 2012. He received the TPC top ranked paper award of IEEE DySPAN 2011. He was also named a Fellow of the School of Graduate Studies, Memorial University of Newfoundland in 2008. His current research interests are wireless sensor networks, cognitive radio networks, software defined networking, the Internet of Things, nanonetworks, and wireless communications in challenged environments.

Min Luo is the head and chief architect of Advanced Networking at Huawei's Shannon (IT) Lab since March 2012. For more than 11 years with IBM, he served as chief/executive architect, SWG's Strategy and Technology, Global Business Solution Center, Industry Solutions, and Center of Excellence for Enterprise Architecture and SOA. He also worked as senior operations research analyst, senior manager, and director of Transportation Network Planning and Technologies for two Fortune 500 companies for seven years. He is certified and was awarded as the Distinguished Lead/Chief Architect from Open Group in 2008. He is an expert in software defined networking; enterprise architecture and SOA; software engineering; and business analytics, intelligence, and optimization. He coauthored two books, including the pioneering *Patterns: Service Oriented Architecture and Web Services*, in 2004. He received his Ph.D. in electrical engineering from Georgia Tech in 1992.

Wu Chou [F] is VP, chief IT scientist, and head of Huawei Shannon (IT) Lab. He is an expert in the field of IT, SDN, data centers, cloud computing, big data, communication, and speech and language processing. He graduated from Stanford University in 1990 with a Ph.D. degree in electrical engineering and worked at leading R&D organizations 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 IT areas. He has published over 150 journal and conference papers, and holds 32 U.S. and international patents with many additional patent applications pending. He received the Bell Laboratories President's Gold Award in 1997, the Avaya Leadership Award in 2005, and the outstanding standard and patent contribution award in 2008 and 2009.