Challenges and Solutions to Consistent Data Plane Update in Software Defined Networks

Sharvari Komajwar
The University of Texas at San Antonio
San Antonio, Texas-78249
Email: sharvari.komajwar@utsa.edu

Turgay Korkmaz
The University of Texas at San Antonio
San Antonio, Texas-78249
Email: turgay.korkmaz@utsa.edu

***Abstract*— Software Defined Networking (SDN) has emerged as a promising paradigm to make network management easier while supporting various applications requiring different guarantees in terms of performance, availability and correctness. In essence, SDN decouples control and data planes such that a logically centralized SDN controller finds the paths based on given requirements, and then accordingly installs the necessary forwarding rules on the distributed SDN switches, which are simple forwarding devices. Due to various reasons such as congestions, failures or policy changes in the network, the SDN controller needs to find new paths and shift the traffic from the original paths to the new paths. However, during this transition, numerous time consuming steps need to be performed such as updating nodes on the new paths. If not carefully coordinated, these updating steps may cause loops, blackholes, and performance degradations. This paper provides a detailed survey on the existing techniques that try to consistently update the new state of data plane while also discussing some future challenges.**

I. Introduction

Software Defined Networking (SDN) has been receiving significant attention due to its promises in simplifying the network management and supporting emerging applications, especially real-time cloud and big-data applications. Unlike traditional networking paradigm, SDN has separated network management and control from forwarding functions. This decoupling of control plane and data plane enables SDN to be more flexible and feasible for many common network services such as routing [28]–[30], multicast [32], [33], security [31], bandwidth management, quality of service (QoS) [16], [34], [35] and storage optimization etc.

In general, SDN has a logically centralized controller (software) which makes network management and control decisions (e.g., finding paths based on current topology and given traffic requirements). The controller then installs the necessary forwarding rules on the simple forwarding switches in the data plane through a standard protocol (e.g., OpenFlow [36]). The behavior of each switch in the data plane is determined by the installed rules. Upon receiving a packet, each switch looks up the rules installed by the controller. If there is a match, the packet is forwarded to the next switch based on the matched rule. Otherwise, the switch informs the controller about this packet and expects to get the necessary new rules from the controller.

SDN has several advantages as a result of decoupling the control plane and data plane. However, it also introduces new challenges such as how to efficiently handle congestion and node/link failure. Many techniques have been proposed to deal with congestion [1], [2] and link/node failure [3], [4]. In general, SDN controller can find new paths for the affected flows using various path selections algorithms proposed in the literature [5]–[9], and switch the traffic from the old paths to the new paths. Unfortunately, the switching from the old paths to the new paths is a very challenging taks due to the distributed nature of the data plane. If it is not done in a consistent manner, it may cause loops and/or blackholes.

In SDN, before switching the traffic from an old path to a new path, the controller needs to install the new rules to the switches on the new path. Clearly, the controller-switch interaction will involve some delay in the update process. Moreover, this delay is not predictable because of the different reaction times of the specific hardware and/or the current load. As a result, it is very difficult to ensure that all the switches are consistently updated before the next packet is sent. Therefore, one of the key challenges in SDN is *how to update the data plane for a new path quickly while not violating the consistency of underlying network*. Accordingly, many techniques have been proposed in the literature [11]–[21]. These techniques differ from each other in terms of their rule placement policies or consistency properties that they try to preserve during the update. Some researchers have also focused on the optimization objectives. In this paper, we give a detailed overview of the existing techniques for consistent data plane update while discussing some future challenges.

This paper is organized as follows. Section II gives the brief introduction of update problem. Section III classifies the techniques proposed in the literature. Section IV discusses open problems that are not addressed yet in the literature. Section V concludes the paper.

II. Network Update Problem In SDN

A dependable SDN network should be able to reconfigure and update the state (i.e., rules) of the underlying switches so that it can cope with various problems such as link/node failures, congestions, or policy changes. Upon the need of switching from an old path to a new path, the SDN controller computes the new path and installs new rules on the corresponding switches. However, due to the distributed structure of SDN switches, it is very difficult to make sure that all the

switches are updated before the next packet arrives. If any one of the switches on the new path is not updated by the controller in a timely manner, that switch may cause a loop in the network or drop the packets due to missing rule. It may also cause performance related problems such as congestions or may violate the policies in the network.

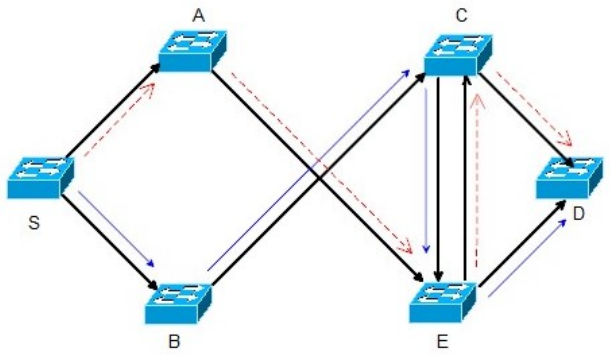

Fig. 1. A simple SDN Topology.

For example, suppose initially all the packets that belong to theflow from S to D are forced to follow the path $\{S, A, E, C, D\}$ shown by the red (dashed) arrows in Fig. 1. Accordingly, SDN controller installs following rules on S, A, E and C: dst(D) $\rightarrow$ fwd(A), dst(D) $\rightarrow$ fwd(E), dst(D) $\rightarrow$ fwd(C) and dst(D) $\rightarrow$ fwd(D), respectively. Suppose, there is a congestion on link $A - E$ at time t and the traffic from S to D needs to be shifted from the old path to the new path $\{S, B, C, E, D\}$ shown by blue (straight) arrows. This transition from the old path to the new path requires SDN controller to install the following new rules on S, B, C and E: dst(D) $\rightarrow$ fwd(B), dst(D) $\rightarrow$ fwd(C), dst(D) $\rightarrow$ fwd(E) and dst(D) $\rightarrow$ fwd(D), respectively. Suppose, at time t + 2 all the switches on the new path get updated and all the packets after that are routed through the new path. But at time t + 1 it is not guaranteed that all the switches are updated or not. In our example, suppose all the switches on the new path except E are updated. So at time t + 1 when packets come to C from B, C will forward them to E based on the new rule installed by the controller. However, since E is not updated yet, it will forward it to C again based on the old rule, resulting in a loop. These packets will loop between C and E till E gets the update from the controller. Clearly, this situation causes unnecessary use of network resources and incurs additional performance penalties (e.g., delay, loss) in the network.

So, one of the challenging problems in the current SDN is how to shift the traffic from an old path to a new path as soon as possible while guaranteeing that there will be no loop or black-hole during this transition. In addition to providing loop-free and black-hole-free update mechanisms, it is necessary to take into account the consistency of the network in terms of policy as well as performance. Accordingly, the researchers have been extensively investigating these issues under the general umbrella of Network State Update Problem. In the next section, we provide an overview of the techniques proposed in the literature.

III. Overview of Data Plane Update Techniques

We classify the existing data plane update techniques in terms of (a) rule placement policies they used, (b) consistency properties they focused on, and (c) the optimization goals they tried to achieve. Table I shows the summary classification of the existing studies. We now describe the datils of these techniques in the following three subsections.

TABLE I
Overview of Update Mechanisms in SDN.

	Loop-free / Blackhole free	
Rule Addition	[10]	Congestion
	[10], [19]	Policy
Rule Replacement	[16], [26]	Congestion
	[22]	Policy
	[11]–[14], [17]	Loop-free / Blackhole free
Hybrid	-	Congestion
	[15], [20]	Policy

A. Rule Placement Policies

Rule placement policies can be classified based on the operations performed on switches as follows:

1) Rule Replacement: Upon computing a new path, the SDN controller tries to install the new rules into the switches on the new path. If some switches are common in the new and old paths, the old rule for a particular flow gets replaced by the new rule. In the literature, many proposed solutions [11]–[14] use this policy. One of the key issues in these solutions is how to guarantee the connectivity consistency (i.e., loop-free and black-hole-free update). The simplest solution would be to update the switches in the reverse order one at a time. In our example, if the controller updates E, C, B and then S, there will be no loop or black-hole. But, this solution requires four rounds of updates and significantly delays the transition from the old path to the new path. The other extreme is to try to update all at once (in one round). However, as the example in previous section shows, this may result in loop if the order of updates is not carefully controlled. So the key issue here is how to minimize the number of rounds (and thus the update delay) by carefully selecting the order of updated switches in each round so that there will be no loop or black-hole during the update. As we discuss in the next sub-section, researchers have proposed various update techniques that can provide connectivity consistency while using rule replacement policy.

2) Rule Addition: In contrast to the Rule Replacement policy, Rule Addition policy allows SDN controller to add new rules to a switch while also keeping the original rules in the switch. For example, while updating the new path in Fig. 1, switch S will maintain both rules (i.e., dst(D) $\rightarrow$ fwd(A) and dst(D) $\rightarrow$ fwd(B)) instead of replacing the old rule. The new rules are initially inactive. Once all the rules are installed, they are activated based on the 2-Phase commit technique [10]. In this technique, packets are tagged on the ingress switch with either "New label" or "Old label". In the internal switches, packets with "New label" are forwarded according to the new rule while the packets with "Old label" are forwarded according to the old rule. This way, packets can be forwarded with either rule. Initially packets are tagged with "Old label". After installing the new rules and confirming their sucussefull installation on all the internal switches using 2-phase commit, the conroller instructs the ingress switch to mark the packets with "New label". In this way, the transition from an old path to new path can be completed with minimum delay and loss.

Though adding a rule minimizes the delay and loss, the wastage of critical network resources (e.g., expensive and rare switch TCAM memory) is an important disadvantage of this technique. Therefore, as we discuss in the next subsection, most of the existing techniques consider rule replacement rather than rule addition. Morever, rule addition technique assumes that the old path is operational while the new one is being established. However, this might not be the case when there is a failure on the old path. Accordingly, a new path must quickly be establised and used instead of the old one.

3) Hybrid Approach: This policy tries to combine both rule replacement and rule addition policies to reduce the memory overhead while mnimizing the number of rounds in updating the switches. For example, Vissicchio et al. presented FLIP [15] which uses the hybrid approach for providing per-packet consistent updates, where each packet follows either the old path or the new path but not the combinations.

B. Consistency Properties

Proposed solutions can also be classified based on consistency properties that they provide. In [17], authors has mentioned wide variety of consistency properties which need to be provided during updates. In the literature, several works have been inspired by the above mentioned 2-phase commit technique. Most of the contributions provide additional guarantee such as congestion free SDN updates [16] or preservation of policy constraints.

1) Connectivity consistency: Network has to guarantee that there exist a working path during the updates. There are two types of connectivity issues need to be handled during updates: (i) Loops: If there is a switch (e.g., C) on the new path which is updated to send the traffic to another switch (e.g., E) which happens to be on the old path sending the traffic to C, then the packets will circulate between $C - E$ till E is updated;
(ii) Black-holes: If there is a switch (e.g., B) on the new path which is not updated yet and receives the traffic, then the packets will get dropped due to missing forwarding rule in this switch.

Loop avoidance is the most basic consistency property and has been intensively studied in the literature. Mahajan and Wattenhofer in [17] first proposed destination based loop-free network update. Instead of adding rules in the switches, they achieved consistent network updates by replacing rules in multiple rounds based on the dependencies. Their solution can be divided into two modules. First module is responsible to generate the plan to update the network. This update plan consists of a directed acyclic graph (DAG) in which rule updates are represented by nodes and directed edges represent dependencies among these rules. Update plan is generated in two steps. In first steps, Dependency graph is built based on old rules, new rules and desired consistency property such as loop-freedom. Second step finally outputs the DAG from the dependency graph by breaking loops. Second module of this solution quickly applies this plan to the current network. In essence, these two modules try to handle two basic concerns related to update problems: consistency and update time (delay). One of the limitations in their solution is that every switch in the network forwards the packet based on its destination. For example, when two flows from different sources to the same destination meet at a common switch, the rest of their paths should be the same.

Ludwig et al. [13] initiated the study of arbitrary route updates which are not destination based. In their technique, update problem can be converted into optimization problem on a directed graph which consists of all the switches on the old and new paths. Nodes which are not common on both paths can be updated trivially, so they can be removed from the graph. Ludwig et al. have introduced two different definitions of loop-free network updates: $Strong\ Loop-freedom\ (SLF)$ and $Relaxed\ Loop-freedom\ (RLF)$. In SLF, forwarding rules installed on all the switches in the network should be loop-free while RLF requires that forwarding rules installed on the switches along the path from source to destination should be loop-free. The main goal in [13] is determine how many communication rounds k are needed to solve the loop-free network update problem. It can easily be decided if the network can be updated in $k = 2$ round or not. But for $k = 3$, the problem is shown to be NP-Hard using 3-SAT reduction. Update algorithms proposed in [13], [17] are mainly designed for a single path [13] or single destination [17] update. In [11], Dudycz et al. proposed a loop-free update algorithm to update multiple flows at the same time while minimizing the number of rounds.

As mentioned above, another consistency property is black-hole free network updates. This property is easy to guarantee by implementing some default matching rule which is never updated. However, it could in turn induce forwarding loops. If there is currently no blackhole for any destination, a straightforward mechanism would be to install new rules with a higher priority, and then delete the old rules [12], [17]. But in the presence of memory limits and guaranteeing loop-freedom, finding the fastest blackhole-free update schedule is shown to

be NP-hard [12], calling for further research.

2) Policy consistency: Network operators often have some high-level requirements and policies such as security, performances etc. To enforce such policies, the network has to route every packet through certain nodes (e.g. Firewall) or sub-paths. When switching from an old path to a new path, the underlying update mechanism should preserve such high-level policies. Accordingly, several policy preserving update mechanims have proposed while each is focusing on some specific policies such as Per Packet Consistency (PPC) and Waypoint Enforcement (WPE).

Per Packet Consistency (PPC) requires every packet to travel either on its initial path or on its new path, but not on the combination of both. This is because both paths are bounded with some high-level policies such as security and/or performance requirements that need to be enforced. In Waypoint Enforcement (WPE), it is not always necessary to follow either the old path or new path. Sometimes it is sufficient to send the traffic through a sub-path or a certain node. WPE generally takes place for security requirements. For example, sometimes to fulfill security requirements all the packets need to traverse a firewall. Here we refer a single node (waypoint) as a firewall. Suppose in the topology given in Fig. 1, switch E is assigned to be a firewall and network administrator impose the policy that any flow traveling from S to D should go through E at anytime. So to maintain policy consistency, updates should be done in such a way that each path during the updates should pass through switch E.

Researchers have propsed various techniques to preserve the PPC and WPE policies during updates. One straightforward solution to preserve PPC is to use the above mentioned 2-Phase commit technique which activates the new path after all the new rules are installed on all the switches along the new path. By taging packets with the "Old label" or "New label," it gurantees that each packet will go through either the old path or the new path, respectively. Unsurprisingly, this approach guarantees per-packet consistency. A major disadvantage of 2-phase commit is that it doubles the consumed memory on switches because of rule addtitions. Other disadvantages include requiring header space, tagging overhead, and complications with middleboxes changing tags. It also requires devices to maintain both the initial and final sets of forwarding rules throughout the update, in order to possibly apply any of the two sets according to packet tags.

To address some of these issues, Katta et al. in [19] have proposed to split the updates into multipe rounds. In each round, the controller determines the consistent sub-updates for the global updates and apply 2-phase commit to it. In this algorithm, as the number of rounds increases, TCAM memory size requirements decreases. But this approach is expensive in terms of update time. In other words, it can limit the memory overhead on each switch at any moment in time but at the price of slowing down the update. In general, the switch-memory consumption of 2-phase commit techniques remains a fundamental limitation of this approach, calling for the exploration of alternatives.

In [20], PPC is ensured using hybrid approach discussed in Section III.A.3. This approach first computes the maximal sequence of rule replacements that preserve PPC, and then apply a restricted 2-phase commit procedure on a subset of (non-ordered) devices and flows.

In [15], Vissicchio et al. focus on the problem of preserving generic policies during SDN updates. For each flow, a policy is defined as a set of paths so that the flow must traverse any of those paths in each intermediate state. The proposed algorithm uses both rule replacements and additions (i.e., packet tagging and tag matching) in the returned operational sequences and during its computation rather than considering the two operations in subsequent steps as in [20].

Both the above works argue that it is always profitable to combine rule replacement and additions, as it reduces the amount of memory overhead while keeping the operational sequence always computable.

In [22], authors mainly focused on preserving WPE during updates with minimum delay. They proposed WayUp, an algorithm which guarantees WPE with minimum number of communication rounds. They also found that it may not be possible to achieve WPE through single point and loop-freedom at the same point.

3) Performance consistency: Performance consistency makes sure that the network resources are available during the updates and not oversubscribed. For example, suppose there are two flows from S to D in Fig. 1 and the capacity of each link is 1 Mbps. The first flow *f1* has bandwidth demand of 0.9 Mbps and goes through the path $\{S, A, E, C, D\}$ while the second flow *f2* has bandwidth demand of 0.2 Mbps and goes through the path $\{S, B, C, E, D\}$. Suppose the network provider changes some policies on both flows such that *f1* and *f2* need to be migrated on paths $\{S, B, C, E, D\}$ and $\{S, A, E, C, D\}$, respectively. In this case, if the new path for any one of the flows is updated before the migration of other flow, there will be some congestion on the old path, causing performance inconsistency in the network.

Many techniques [10], [16], [21], [26] consider the traffic volumes and the corresponding constraints raised by the limited capacity of network links. Their goal is to follow these constraints during each step of updates. In the above example, if we consider that both flows are splittable, we can solve this problem in polynomial time by sending 0.5 Mbps of *f1* on original path and remaining flow on new path. But for unsplittable flows this is a NP-Hard problem.

Current algorithms to maintain performance consistency are based on the work by Reitblatt et al. [10]. In [16], Hong et al. have introduced a new system called SWAN which helps to update a network in congestion-free manner and increases the overall utilization of network. Initially it assumes that there are three classes of traffic: Interactive, Elastic and Background. It is observed that it is impossible to update network without congestion if all links are full. SWAN assigns a scratch capacity s at each link. Then, they prove that it is possible to update the network without any congestion in at most *[1/s]-1* steps. Using LP formulation, SWAN also finds

the sequence of minimal number of steps, if exists. Instead of wasting scratch capacity, it allocates it to background traffic.

This LP-formulation was extended by Zheng et al. [21] to include unsplittable flows as well. Furthermore, using randomized rounding with an LP, Zheng et al. can approximate the minimum congestion that will occur if the migration has to be performed using minimal number of updates. Also in [26], Wang et al. divided new routing path into multiple independent segments and identified critical nodes which shift the flow from the old path to the new path. Instead of building global dependency graph, they built a local dependency graph with potentially congested links using critical nodes and then proposed a heuristic algorithm to resolve those dependencies in it.

In [24], Xu et al. have introduced a new aspect of network update problem. They argue that network update delay depends on not only update scheduling but also how many flows controller will update and which paths will be updated. Another challenge during network update is how to keep track of the current network configuration as the network state changes frequently. If the workload of network is changing frequently and update delay is significant, then the new configuration may be no longer efficient for the workload after update. To addresse the real-time route update, Xu et al. have considered the optimization of flow route selection and update scheduling jointly. They formulated the delay-satisfied route update (DSRU) problem with three constraints. First constraint is *congestion-free constraint* that guarantees no congestion during the update process. Second constraint is *route-consistency constraint* in which each packet should be forwarded either with the new configuration or the old configuration and not with the mixture of two. Third constraint is *real-time constraint* which guarantees that the maximum delay of route update on all switches should not exceed some tolerated delay. Further they proved its NP-Hardness by reducing from multi-commodity flow problem (MCF) and proposed two algorithms: greedy and an approximation algorithm based on randomized rounding method.

Mizrahi et al. in [27] have proposed a unique solution for the congestion during updates. They argue that sometimes two or more switches need to be updated at the same time to avoid congestion. Their solution is specially applicable for the situation where flow swapping occurs as a part of load balancing. First, they proved that in some scenarios flows are need to be swapped. It is inevitable. Then they provided a mechanism to apply updates on switches almost at the same time. This mechanism is based on OpenFlow [36] *Bundle* feature (Bundle is a sequence commands applied as a single operation). They proposed *Scheduled Bundle*, to apply all the commands in specific pre-determined time. This feature allows all the switches to update the new path within a specific time limit.

C. Optimization Goals

Proposed techniques can be classified based on different types of objectives that need to be achieved during updates such as maximizing the number of switches updated in a single round without violating any consistency, minimizing the communication amount and delay between controller and switches or minimizing the extent to which link capacities are oversubscribed during the update.

In [18], Zheng et al. proposed an algorithm to minimize the congestion that occurs during network updates. They argue that for production scale networks (where the number of flows is more than 5K), existing solutions [16] are too slow since they require to solve a series of LPs to calculate the congestion-free update. Since many low priority flows can tolerate packet loss due to congestion, Zheng et al. proposed congestion-minimizing network update instead of congestion-free network update. They proposed two optimization problems: minimum congestion update problem (MCUP) and bounded congestion update problem (BCUP). MCUP aims at minimizing congestion during updates within x intermediate steps where x is given. Where BCUP aims to minimize the number of intermediate steps and corresponding routing at each step such that maximum congestion will be less than threshold y. They further proved that these two problems are NP-hard and provided approximation algorithms and heuristics for that.

IV. Discussion

In the previous section, we discussed various techniques that have been proposed in the literature to solve consistent update problem in SDN. The main goal of all these techniques is to shift the traffic from an old path to a new path while avoiding loops and blackholes in the network. To ahieve this goal, it is necessary to update the state of the network in a consistent manner. The existing solutions can in general be classified into two types: 2-phase commit-based techniques [10] and ordered round-based techniques [12], [17]. Both types have certain advantages and disadvantages. For example, 2-phase commit-based techniques can easily maintain all types of consistencies mentioned in the previous section. But the major challenge faced by these techniques is the excessive use of memory. Morever, since these techniques apply a new rule only when the controller receives confirmation from every switch, a significant delay may be incurred in case one of the switches responds slowly.

To mitigate some of the problems in 2-phase commit-based techniques, many researchers considered round-based techniques. While round-based techniques do not need excessive memory, they still face the challenge regarding delay. As the number of rounds increases, the time required to update the new path increases. In most of the round-based techniques, the number of rounds are determined based on the dependencies of rules. Also time required for each round is dependent on the communication between each switch and the controller. In the worst case, the number of rounds can be equal to the number of switches involved in the path switching. So there is still need for efficient techniques that can update network in minimum rounds without excessive memory consumption.

All of the above techniques assume that the controller has already calculated the new path as a backup path for particular flow. In case of events such as congestion/failure, the controller updates the new path. That means the way controller calculates the new path may affect the overall delay required to update the network. Many path finding algorithms are proposed in the literature [5]–[9]. They can be divided into two classes. First class assumes that controller calculates two paths for each flow and install both paths on the switches, one as primary and other one as a backup path. Using this technique, the network can be updated quickly and efficiently. However, installing two paths for each flow is a wastage of critical network resources such as switch TCAM memory. Moreover, the backup path may not be available when needed. So a new path needs to be computed on demand.

In case of second class of path finding algorithms, the controller calculates a totally new path from source to destination upon detecting congestion/failure. In this case, the total time required to update the network includes time required to calculate the new path and time to update switches on the new path. In the worst case, if the new path is totally different from the old path and contain many critical switches (i.e., Switches which have entries for both old and new paths with different interfaces), the probability of creating loop/blackhole during update may increase. Apparently, this will increase delay and overhead during the update. One solution is to find an optimum path which has maximum number of overlapping switches (i.e., Switches which have entry for both new and old paths with same interface) to minimize the time and overhead required to update the network. So there is a need for developing new path selection algorithms that can simultaneously find the optimum path on demand and update the network in a timely and efficient manner.

In case of computing paths on demand, the controller needs to have the accurate state information of the underlying data plane. However, due to the distrubuted nature of the data plane, the accurate state information cannot easily be otained. For high accuracy, the conroller needs to query the switches frequently or the switches should send every change to the controller. Unfortunately, this will incur significant protocol overhead. So there is a significant need for designing efficient mechanisms to provide accurate state information to the controllers and/or new path selection algorithms that can take inaccurate state information into account.

Last but not least, many of the existing techniques [10], [19] assume that the old path is still available to carry packets while switching the traffic over a new path. While this assumption is true in case of shifting traffic becasue of policy changes or congestion, it is clearly not true in case of link/node failures. Since all the packets will be dropped at the point of failure, the path shifting should be done in a much more timely manner. In [25], Huang et al. have considered this problem first time and proposed source routing based protocol to handle failures. However, more work still needs to be done for develeoping locally and globally efficient mechanisms to cope with unpredictable events such as link/node failures.

V. Conclusions

In this paper, we provided the overview of the techniques used to solve the network update problems along with their classification according to Rule placement policy they used, type of consistency they focus on and performance goals they try to achieve. We also presented some open challenges that require further research.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong et al., B4: Experience with a globally-deployed software defined wan, in SIGCOMM, 2013.

[2] S. Agarwal, M. Kodialam, and T. Lakshman, Traffic engineering in software defined networks, in INFOCOM, 2013.

[3] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, Congestion-aware single link failure recovery in hybrid sdn networks, in INFOCOM, 2015.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang et al., Hedera: Dynamic flow scheduling for data center networks. in NSDI, 2010.

[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula et al., Devoflow: Scaling flow management for high-performance networks, in SIGCOMM, 2011.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang, Microte: Fine grained traffic engineering for data centers, in CoNEXT, 2011.

[7] M. Suchara, D. Xu, R. Doverspike, D. Johnson et al., Network architecture for joint failure recovery and traffic engineering, in SIGMETRICS, 2011.

[8] Y. D. Lin, H. Y. Teng, C. R. Hsu, C. C. Liao and Y. C. Lai, "Fast failover and switchover for link failures and congestion in software defined networks," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, 2016.

[9] A. Capone, C. Cascone, A. Q. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in sdn with openstate," in Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the. IEEE, 2015, pp. 2532.

[10] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In Proc. ACM SIGCOMM, pages 323334, 2012.

[11] S. Dudycz, A. Ludwig, and S. Schmid. Can't touch this: Consistent network updates for multiple policies. In Proc. IEEE/IFIP DSN, 2016.

[12] K.-T. Frster, R. Mahajan, and R. Wattenhofer. Consistent Updates in Software Defined Networks: On Dependencies, Loop Freedom, and Blackholes. In Proc. IFIP Networking, 2016.

[13] A. Ludwig, J. Marcinkowski, and S. Schmid. Scheduling loop-free network updates: Its good to relax! In Proc. ACM PODC, 2015.

[14] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure. Lossless migrations of link-state igps. IEEE/ACM Transactions on Networking (TON), 20(6):18421855, 2012.

[15] S. Vissicchio and L. Cittadini. FLIP the (Flow) Table: Fast LIghtweight Policy-preserving SDN Updates. In Proc. IEEE INFOCOM, 2016.

[16] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. SIGCOMM Comput. Commun. Rev., 43(4):1526, 2013.

[17] R. Mahajan and R. Wattenhofer. On Consistent Updates in Software Defined Networks. In Proc. ACM HotNets, 2013.

[18] J. Zheng; H. Xu; G. Chen; H. Dai; J. Wu, "Congestion-Minimizing Network Update in Data Centers," in IEEE Transactions on Services Computing , vol.PP, no.99, pp.1-1 doi: 10.1109/TSC.2016.2631519

[19] N. P. Katta, J. Rexford, and D. Walker. Incremental consistent updates. In Proc. ACM SIGCOMM HotSDN, 2013.

[20] S. Vissicchio, L. Vanbever, L. Cittadini, G. Xie, and O. Bonaventure. Safe Update of Hybrid SDN Networks. Technical report, UCLouvain, 2013.

[21] J. Zheng, H. Xu, G. Chen, and H. Dai. Minimizing transient congestion during network update in data centers. In Proc. ICNP, 2015.

[22] A. Ludwig, M. Rost, D. Foucard, and S. Schmid. Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies. In Proc. ACM HotNets, 2014.

[23] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid. Transiently secure network updates. In Proc. ACM SIGMETRICS, 2016.

[24] Hongli Xu, Zhuolong Yu, Xiang-Yang Li, Chen Qian, Liusheng Huang and Taeho Jung, "Real-time update with joint optimization of route selection and update scheduling for SDNs," 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 2016, pp. 1-10.
[25] Huang Liaoruo, Shen Qingguo and Shao Wenjuan, "A source routing based link protection method for link failure in SDN," 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2016, pp. 2588-2594.
[26] W. Wang, W. He, J. Su, and Y. Chen. Cupid: Congestion-free consistent data plane update in software defined networks. In Proc. IEEE INFOCOM, 2016.
[27] T. Mizrahi and Y. Moses, "Software defined networks: It's about time," IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, 2016, pp. 1-9.
[28] L. W. Cheng and S. Y. Wang, "Application-Aware SDN Routing for Big Data Networking," 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, 2015
[29] C. Chen, B. Li, D. Lin and B. Li, "Software-defined inter-domain routing revisited," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, 2016
[30] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, Sdx: A software defined internet exchange, in Proc. ACM SIGCOMM, 2014
[31] Q. Yan, F. R. Yu, Q. Gong and J. Li, "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 602-622, Firstquarter 2016.
[32] W. Cui and C. Qian, "Scalable and Load-Balanced Data Center Multicast," 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, 2015
[33] J. Ruckert, J. Blendin, R. Hark and D. Hausheer, "DYNSDM: Dynamic and flexible software-defined multicast for ISP environments," 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, 2015
[34] J. M. Wang; Y. Wang; X. Dai; B. Bensaou, "SDN-based Multi-Class QoS Guarantee in Inter-Data Center Communications," in IEEE Transactions on Cloud Computing , vol.PP, no.99, pp.1-1 doi: 10.1109/TCC.2015.2491930
[35] A. Ghosh, S. Ha, E. Crabbe, and J. Rexford, Scalable multi-class traffic management in data center backbone networks, Selected Areas in Communications, IEEE Journal on, vol. 31, no. 12, 2013.
[36] Open Networking Foundation, OpenFlow Switch Specification, Version 1.4.0, 2013.