A

Survey of Consistent Network Updates

Klaus-Tycho Foerster, ETH Zurich, Switzerland
Stefan Schmid, Aalborg University, Denmark
Stefano Vissicchio, Universite catholique de Louvain, Belgium

Computer networks have become a critical infrastructure. As such, networks should not only meet strict requirements in terms of correctness, availability, and performance, but they should also be very flexible and support fast updates, e.g., due to a change in the security policy, increasing traffic, or failures.

In this paper, we present a structured survey of mechanisms and protocols to update computer networks in a fast and consistent manner. In particular, we identify and discuss the different desirable consistency properties that should be provided throughout a network update, the algorithmic techniques which are needed to meet these consistency properties, and the implications on the speed and costs at which updates can be performed. We also explain the relationship between consistent network update problems and classic algorithmic optimization ones. While our survey is mainly motivated by the advent of Software-Defined Networks (SDNs) and their primary need for correct and efficient update techniques, the fundamental underlying problems are not new, and we provide a historical perspective of the subject as well.

CCS Concepts: •**General and reference** → **Surveys and overviews;** •**Networks** → **Network algorithms;** •**Computer systems organization** → **Distributed architectures;**

Additional Key Words and Phrases: Network Updates, Algorithms, Software-Defined Networks, OpenFlow, TCAMs

1. INTRODUCTION

Computer networks such as datacenter networks, enterprise networks, carrier networks etc. have become a critical infrastructure of the information society. The importance of computer networks and the resulting strict requirements in terms of availability, performance, and correctness however stand in contrast to today's ossified computer networks: the techniques and methodologies used to build, manage, and debug computer networks are largely the same as those used in 1996 [Barefoot Networks 2016]. Indeed, operating traditional computer networks is often a cumbersome and error-prone task, and even tech-savvy companies such as GitHub, Amazon, GoDaddy, etc. frequently report issues with their network, due to misconfigurations, e.g., resulting in forwarding loops [Imbriaco 2012; Jackson 2012; Mohan 2011; United 2011]. An anecdote reported in [Barefoot Networks 2016] illustrating the problem, is the one by a Wall Street investment bank: due to a datacenter outage, the bank was suddenly losing millions of dollars per minute. Quickly the compute and storage emergency teams compiled a wealth of information giving insights into what might have happened. In contrast, the networking team only had very primitive connectivity testing tools such as ping and traceroute, to debug the problem. They could not provide any insights into the actual problems of the switches or the congestion experienced by individual packets, nor could the team create any meaningful experiments to identify, quarantine and resolve the problem [Barefoot Networks 2016].

Software-defined networking is an interesting new paradigm which allows to operate and verify networks in a more principled and formal manner, while also introducing flexibilities and programmability, and hence foster innovations. In a nutshell, a Software-Defined Network (SDN) outsources and consolidates the control over the forwarding or routing devices (located in the so-called *data plane*) to a logically centralized controller software (located in the so-called *control plane*). This decoupling allows to evolve and innovate the control plane independently from the hardware constraints of the data plane. Moreover, OpenFlow, the de facto standard SDN protocol today, is based on a simple match-action paradigm: the behavior of an OpenFlow switch is defined by a set of forwarding rules installed by the controller. Each rule consists of a match and an action part: all packets matched by a given rule are subject to the corresponding action. Matches are defined over Layer-2 to Layer-4 header fields (e.g., MAC and IP addresses, TCP ports, etc.), and actions typically describe operations such as

Author's addresses: Klaus-Tycho Foerster, (Current address) Microsoft Research, Redmond, WA, USA; Stefan Schmid, Aalborg University, Aalborg, Denmark; Stefano Vissicchio, (Current address) University College London, London, UK.

forward to a specific port, drop, or update certain header fields. In other words, in an SDN/OpenFlow network, network devices become simpler: their behavior is defined by a set of rules installed by the controller. This enables formal reasoning and verification, as well as flexible network update, from a logically centralized perspective [Kazemian et al. 2012; Khurshid et al. 2012]. Moreover, as rules can be defined over multiple OSI layers, the distinction between switches and routers (and even simple middleboxes [Feamster et al. 2014]) becomes blurry.

However, the decoupling of the control plane from the data plane also introduces new challenges. In particular, the switches and controllers as well as their interconnecting network form a complex asynchronous distributed system. For example, a remote controller may learn about and react to network events slower (or not at all) than a hardware device in the data plane: given a delayed and inconsistent view, a controller (and accordingly the network) may behave in an undesirable way. Similarly, new rules or rule updates communicated from the controller(s) to the switch(es) may take effect in a delayed and asynchronous manner: not only because these updates have to be transmitted from the controller to the switches over the network, but also the reaction time of the switches themselves may differ (depending on the specific hardware, data structures, or concurrent load).

Thus, while SDN offers great opportunities to operate a network in a correct and verifiable manner, there remains a fundamental challenge of how to deal with the asynchrony inherent in the communication channel between controller and switches as well as in the switches themselves. Accordingly, the question of how to update network behavior and configurations correctly yet efficiently has been studied intensively over the last years. However, the notions of correctness and efficiency significantly differ across the literature. Indeed, what kind of correctness is needed and which performance aspects are most critical often depends on the context: in security-critical networks, a very strong notion of correctness may be needed, even if it comes at a high performance cost; in other situations, however, short transient inconsistencies may be acceptable, as long as at least some more basic consistency guarantees are provided (e.g., loop-freedom).

We observe that not only is the number of research results in the area growing very quickly, but also the number of models, the different notions of consistency and optimization objectives, as well as the algorithmic techniques: Thus, it has become difficult to keep an overview of the field even for active researchers. Moreover, many of the underlying update problems are not entirely new or specific to SDN: rather, techniques to consistently update legacy networks have been studied in the literature, although they are based on the (more restrictive) primitives available in traditional protocols (e.g., IGP weights).

We therefore believe that it is time for a comprehensive survey of the subject.

1.1. The Network Update Problem

Any dependable network does not only need to maintain a range of static invariants, related to correctness, availability, and performance, but also needs to be flexible and support reconfigurations and updates. Reasons for updating a network are manifold, including:

(1) *Change in the security policy:* Due to a change in the enterprise security policy, traffic from one subnetwork may have to be rerouted via a firewall before entering another subnetwork. Or, in the wide-area network, the set of countries via which it is safe to route sensitive traffic may change over time.
(2) *Traffic engineering:* In order to improve traffic engineering metrics (e.g., minimizing the maximal link load), a system administrator or operator may decide to reroute (parts of) the traffic along different links. For example, many Internet Service Providers switch between multiple routing patterns during the day, depending on the expected load. These patterns may be precomputed offline, or may be computed as a reaction to an external change (e.g., due to a policy change of a Content Distribution Provider).
(3) *Maintenance work:* Also maintenance work may require the update of network routes. For example, in order to replace a faulty router, or to upgrade an existing router, it can be necessary to temporarily reroute traffic.

(4) *Link and node failures:* Failures happen quite frequently and unexpectedly in today's computer networks, and typically require a fast reaction. Accordingly, fast network monitoring and update mechanisms are required to react to such failures, e.g., by determining a failover path.

Despite these changes, it is often desirable that the network maintains certain consistency properties *throughout the update*. Those properties may include per-packet path consistency (a packet should be forwarded along the old or the new route, but never a mixture of both), waypoint enforcement (a packet should never bypass a firewall), or at least correct packet delivery (at no point in time packets should be dropped or trapped in a loop).

While the above reasons for network updates are general and relevant independently of the adopted paradigm, the decoupling of control- and data-plane, as well as the flexibility allowed by the SDN architecture are likely to increase the frequency of network updates, e.g., for supporting more fine-grained and frequent optimization of traffic paths [Jain et al. 2013].

1.2. Our Contributions

This paper presents a comprehensive survey of the consistent network update problem.

We identify and compare both the different notions of consistency and the different performance objectives considered in the literature. In particular, we provide an overview of the algorithmic techniques required to solve specific classes of network update problems, and discuss inherent limitations and tradeoffs between the achievable level of consistency and the speed at which networks can be updated. In fact, as we will see, some update techniques are not only less efficient than others, but with them, it can even be impossible to consistently update a network.

We also present a historical perspective, surveying the consistency notions provided in traditional networks and discussing the corresponding techniques accordingly.

Moreover, we put the algorithmic problems into perspective and discuss how these problems relate to classic optimization and graph theory problems, such as multi-commodity flow problems or maximum acyclic subgraph problems.

The goal of our survey is to (1) provide active researchers in the field with an overview of the state-of-the-art literature, but also to (2) help researchers who only recently became interested in the subject to bootstrap and learn about open research questions.

1.3. Paper Organization

The remainder of this paper is organized as follows. §2 presents a historical perspective and reviews notions of consistency and techniques both in traditional computer networks as well as in Software-Defined Networks. §3 then presents a classification and taxonomy of the different variants of the consistent network update problems. §4, §5, and §6 review models and techniques for connectivity consistency, policy consistency, and performance consistency related problems, respectively. §7 discusses proposals to further relax consistency guarantees by introducing tighter synchronization. In §8, we identify practical challenges. After highlighting future research directions in §9, we conclude our paper in §10.

2. THE NETWORK UPDATE PROBLEM FROM THE ORIGINS TO SDN

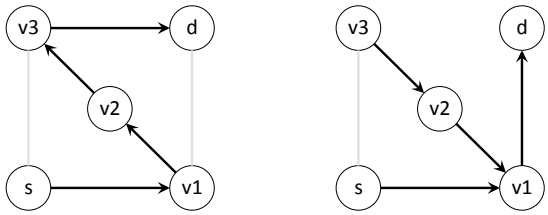
Any computer network needs to provide basic mechanisms and protocols to set and change forwarding rules and paths. Unsurprisingly, the study of consistent network updates is therefore not introduced by SDN. For example, a forwarding loop can quickly deplete switch buffers and harm the availability and connectivity provided by a network, and protocols such as the Spanning Tree Protocol (STP) have been developed to ensure loop-free layer-2 forwarding at any time. However, consistency problems may also arise on higher layers in the OSI stack.

In this section, we provide a historical perspective on the many research contributions that lately focused on guaranteeing consistency properties during network updates, that is, while changing packet-processing rules on network devices.

We first discuss update problems and techniques in traditional networks (§2.1-2.2). In those networks, forwarding rules are computed by routing protocols that run standardly-defined distributed algorithms, whose output is influenced by both physical topology (e.g., active links) and routing configurations (e.g., logical link costs). Pioneering update works aimed at avoiding transient inconsistencies due to modified topology or configurations, mainly focusing on the Interior Gateway Protocols (IGPs) that are commonly used to control forwarding within a single network. A first set of contributions tried to modify IGP protocol definitions, mainly to provide forwarding consistency guarantees upon link or node failures. Progressively, the research focus has shifted to a more general problem of finding sequences of IGP configuration changes that modify forwarding while guaranteeing forwarding consistency, e.g., for service continuity (§2.1). More recent works have also considered reconfigurations of protocols different or deployed in addition to IGPs, mostly generalizing previous techniques while keep focusing on forwarding consistency (§2.2).

Subsequently (§2.3), we discuss update problems in the context of SDNs. Those networks are based on a clear separation between controller (implementing the control logic) and dataplane elements (applying controller's decision on packets). This separation indisputably provides new flexibility and opens new network design patterns, for example, enabling security requirements to be implemented by careful path computation (done by the centralized controller). This also pushed network update techniques to consider additional consistency properties like policies and performance.

Throughout the section, we rely on the generic example shown in Fig. 1 for illustration. The figure shows the intended forwarding changes to be applied for a generic network update. Observe that possible forwarding loops can occur when we update nodes one by one, because edges (v_1, v_2) and (v_2, v_3) are traversed in opposite directions before and after the update.



(a) Surpassed state (b) Down state

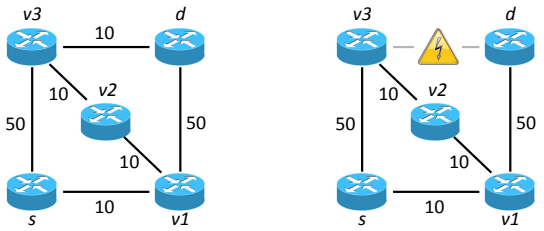
Fig. 1. A network update example, where forwarding paths have to be changed from the Surpassed (Fig. 1a) to the Down (Fig. 1b) state. Arrows represent paths on which traffic (e.g., from s to d) is forwarded, while (gray) undirected edges between nodes represent unused links.

2.1. IGP Reconfigurations

In traditional (non-SDN) networks, forwarding paths are computed by distributed routing protocols. Among them, link-state IGPs are typically used to compute forwarding paths within a network owned by the same administrative entity. They are based on computing shortest-paths on a logical view of the network, that is, a weighted graph which is shared across routers. Parameters influencing IGP computations, like link weights, can be set by operators by editing router configurations.

As an illustration, Fig. 2 shows a possible IGP implementation for the network states shown in Fig. 1. In particular, Fig. 2 reports the IGP graph (consistent with the physical network topology) with explicit mention of the configured link weights. Based on those weights, for each destination (e.g., d in this example), all routers independently compute the shortest paths, and forward the corresponding packets to the next-hops on those paths. Consequently, the IGP configurations in Figs. 2a and 2b respectively produce the forwarding paths depicted in Figs. 1a and 1b.

When the IGP graph is modified (e.g., because of a link failure, a link-weight change or a router restart), messages are propagated by the IGP itself from node to node, so that all nodes rebuild a consistent view of the network: This process is called *IGP convergence*. However, IGPs do not



(a) Surpassed state (b) Down state

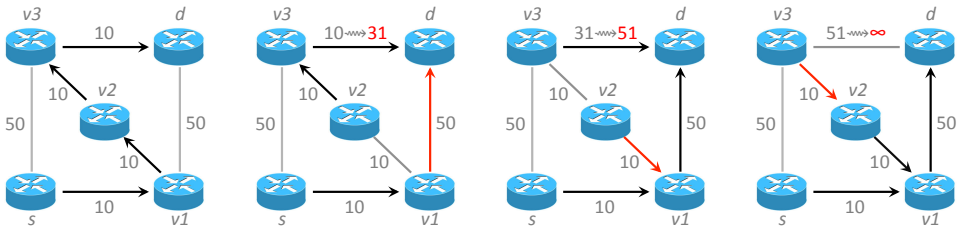
Fig. 2. Possible implementation of pre- and post-update forwarding paths for the update in Fig. 1 in a traditional, IGP-based network. Numbers close to network links represent the corresponding IGP weights.

provide any guarantee on the timing and ordering in which nodes receive messages about the new IGP graphs. This potentially triggers transient forwarding disruptions due to temporary state inconsistency between a set of routers. For example, assume that we simply remove link (v_3, d) from the IGP graph shown in Fig. 2a. This will eventually lead us to the configuration presented in Fig. 2b. Before the final state is reached, the notification that (v_3, d) is removed has to be propagated to all routers. If v_3 receives such notification before v_2 (e.g., because closer to the removed link), then v_3 would recompute its next-hop based on the new information, and starts forwarding packets for d to v_2 (see Fig. 1b). Nevertheless, v_2 keeps forwarding packets to v_1 as it considers that (v_3, d) is still up. This creates a loop between v_3 and v_2: The loop remains until v_2 is notified about the removed link. A similar loop can occur between v_2 and v_1.

Guaranteeing disruption-free IGP operations has been considered by research and industry since almost two decades. We now briefly report on the main proposals in the area.

Disruption-free IGPs have been studied. Early contributions focused on modifying the routing protocols, mainly to avoid forwarding inconsistencies. Among them, protocol extensions have been proposed [Moy et al. 2003; Shaikh et al. 2006; Shand and Ginsberg 2008] to gracefully restart a routing process, that is, to avoid forwarding disruptions (e.g., blackholes) during the software upgrade of a router. Other works focused on preserving forwarding consistency, that is, avoiding loops, upon network failures. For example, François and Bonaventure [2007] propose oFIB, an IGP extension that guarantees the absence of forwarding loops after topological changes (link/node addition or removal). The key intuition behind oFIB is to use explicit synchronization between routers in order to constrain the order in which each node changes its forwarding entries. In particular, each router (say, v_2 in our example) is forced not to update its forwarding entry for a given destination (d in our example) until all its final next-hops (v_1) use their own final next-hops (d in our case). Fu et al. [2008] generalize the previous approach by defining a loop-free ordering of IGP-entry updates for arbitrary forwarding changes. Moreover, PLSN [Shand and Bryant 2010] specializes oFIB: It allows routers to dynamically avoid loops by locally delaying forwarding changes that are not safe. A variant of oFIB, studied by Shi et al. [2009], also extends the reconfiguration mechanism to consider traffic congestion in addition to forwarding consistency.

Modifying protocol specifications may seem the most straightforward solution to deal with reconfigurations in traditional networks, but it actually has practical limitations. First, this approach cannot accommodate custom reconfiguration objectives. For instance, ordered forwarding changes generally work only on a per-destination basis [François and Bonaventure 2007], which can make the reconfiguration process slow if many destinations are involved – while one operational objective could be to exit transient states as quickly as possible. Second, protocol modifications are targeted to specific reconfiguration cases (e.g., single-link failures), since it is intrinsically hard to predict the impact of any possible configuration change on forwarding paths. Finally, protocol extensions are not easy to implement in practice, because of the reluctance of vendors to change their proprietary router software, as well as the additional complexity and potential overhead (e.g., load) induced on routers.



(a) Initial (b) Step 1 (c) Step 2 (d) Final

Fig. 3. Illustration of how intermediate IGP weights (for link (v_3, d) in this case) enable to achieve a loop-free reconfiguration for the update example in Fig. 1.

Limited practicality of protocol modifications quickly motivated new approaches, based on coordinating operations in order to eventually replace the initial configuration with the final one on all network nodes, while guaranteeing absence of disruptions throughout the process. Those approaches, summarized in the following, mainly focused on support for planned operations.

Optimization algorithms can minimize disruptions. As a first attempt, Keralapura et al. [2006] studied the problem of finding the optimal way in which devices and links can be added to a network to minimize disruptions. Many following contributions focused on more fine-grained operations to gain additional degrees of freedom in IGP reconfiguration problems.

A natural choice among finer-grained operations readily supported by traditional routers is tweaking IGP link weights. For example, in [2009] and [2011], Raza et al. propose a theoretical framework to formalize the problem of minimizing a certain disruption function (e.g., link congestion) when the link weights have to be changed. The authors also propose a heuristic to find an ordering in which to modify several IGP weights within a network, so that the number of disruptions is minimal.

While easily applicable to real reconfiguration cases, those approaches assume primitives which are quite coarse grained (e.g., addition of a link, or weight changes), and cannot guarantee the absence of disruptions in several cases: The scenario in Fig. 2 is an example where coarse-grained operations (link removal) cannot prevent forwarding loops.

Progressively changing link weights can avoid loops. Intermediate link weights can be used to avoid disruptions during a reconfiguration – at the cost of slowing down the process. Consider again the example in Fig. 2, and let the final weight for link (v_3, d) conventionally be ∞. In this case, the forwarding loops potentially triggered by the IGP reconfiguration can be provably prevented by using two intermediate weights for link (v_3, d), as illustrated in Fig. 3. The first of those intermediate weights (see Fig. 3b) is used to force v_1 and only v_1 to change its next-hop, from v_2 to d: Intuitively, this prevents the loop between v_2 and v_1. The second intermediate weight (see Fig. 3c) similarly guarantees that the loop between v_3 and v_2 is avoided, i.e., by forcing v_2 to use its final next-hop before v_3. Of course, computing intermediate weights that guarantee the absence of disruptions becomes trickier when multiple destinations are involved.

Such a technique can be straightforwardly applied to real routers. For example, an operator can progressively change the weight of (v_3, d) to 31 by editing the configuration of v_3 and d, then check that the all IGP routers have converged on the paths in Fig. 3b, repeat similar operations to reach the state in Fig. 3c, and finally remove the link safely. Even better, François et al. [2007b] have proved that it is always possible to compute a sequence of intermediate link weights that provably avoids all transient loops when a single link has to be reweighted. Obviously, the weight of multiple links can be changed in a loop-free way, by safely reweighting links one by one.

Additional research contributions focused on minimizing the number of intermediate weights that ensure loop-free reconfigurations. Surprisingly, the problem is *not* computationally hard, despite the fact that all destinations have potentially to be taken into account when changing link weights. Polynomial-time algorithms have been proposed to support planned operations at the per-link [Clad

et al. 2014; François et al. 2007b] (e.g., single-link reweighting) and at a per-router [Clad et al. 2013; Clad et al. 2015] (e.g., router shutdown/addition) granularity.

Ships-in-the-Night (SITN) techniques generalize the idea of incremental changes to avoid loops. To improve the update speed in the case of many link changes and to deal with additional reconfiguration scenarios (from changing routing parameters to replacing an IGP with another), both industrial best practices and research works often rely on a technique commonly called Ships-in-the-Night [Herrero and van der Ven 2010]. This technique builds upon the capability of traditional routers to run multiple routing processes at the same time. Thanks to this capability, both the initial and final configurations can be installed (as different routing processes) on all nodes at the same time. When multiple configurations are installed on the same node, only one of them is preferred and used. Fig. 4 shows the setup for a Ships-in-the-Night reconfiguration for the reconfiguration case in Fig. 2.

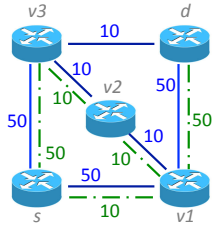


Fig. 4. Ships-in-the-Night setup: All routers run two routing processes, one with the initial configuration (blue, solid lines) and the other with the final configuration (green, dashed and dotted segments).

In SITN, the reconfiguration process then consists in swapping the preference between the initial and the final configurations on every node, typically one by one. Configuration preference can be swapped at a per-destination granularity. This means that (1) for each destination, every node either forwards packets to its initial next-hops or its final ones; (2) at any time during the reconfiguration, distinct nodes can use different configurations; hence, (3) inconsistencies may arise from the mismatch between the configurations used by distinct nodes.

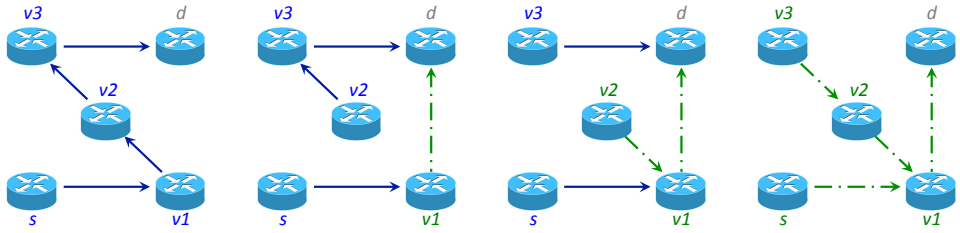


Fig. 5. Illustration of a Ships-in-the-Night reconfiguration that mimicks the progressive link reweighting shown in Fig. 3. In each figure, the colors of router names indicate their respective control-plane preferences at the represented reconfiguration step.

Because of those potential inconsistencies, the Ships-in-the-Night approach opens a new algorithmic problem, that is, to decide a safe order in which to swap preferences on a per-router basis. For example, if the configuration preference is swapped on v_3 before doing the same on v_2 in Fig. 4, we end up with a loop between v_2 and v_3. In contrast, Fig. 5 shows a SITN-based safe reconfiguration that mimics the progressive link weight increment depicted in Fig. 3.

To guarantee the absence of disruptions, configuration preference must then be swapped incrementally, in a carefully-computed order [Vanbever et al. 2011]. Indeed, naive approaches in swapping configuration preferences cannot guarantee disruption-free reconfigurations. For example, replacing

the initial configuration with the final one on all nodes at once provides no guarantee on the order in which new preferences are applied by nodes, hence potentially triggering packet losses and service disruptions (in addition to massive control-plane message storms). Even worse, such an approach could leave the network in an inconsistent, disrupted and hard-to-troubleshoot state if any reconfiguration command is lost or significantly delayed. Similarly, industrial best practices (e.g., [Herrero and van der Ven 2010; Pepelnjak 2007]) only provide rules of thumb which do not apply in the general case, and do not guarantee lossless reconfiguration processes.

The SITN algorithmic problem called for new research contributions. In [Vanbever et al. 2011; 2012], algorithms (based on Linear Programming, and heuristics) have been proposed to deal with several IGP reconfiguration scenarios, including the simultaneous change of multiple link weights, the modification of other parameters (e.g., OSPF areas) influencing IGP decisions, and the replacement of one IGP protocol with another (e.g., OSPF with IS-IS). To minimize the update time, the proposed algorithms try to change the configuration of each router only once, i.e., modifying its forwarding next-hops to all possible destinations altogether. As such, they also generalize the algorithms behind protocol-modification techniques, especially oFIB [François and Bonaventure 2007], that restrict to per-destination operational orderings. Beyond providing ordering algorithms, [Vanbever et al. 2011; 2012] also describe comprehensive system to carry out loop-free IGP reconfigurations in automatically or semi-automatically, i.e., possibly waiting the input from the operator to perform the next set of operations in the computed operational sequence.

2.2. Generalized Routing Reconfigurations in Traditional Networks

Research contributions have been devoted to reconfigurations in more realistic settings, including other protocols in addition to an IGP.

Enterprise networks, with several routing domains. As a first example, the Ships-in-the-Night framework has been used to carry out IGP reconfigurations in enterprise networks. Those networks typically use *route redistribution* [Le et al. 2008], a mechanism enabling the propagation of information from one routing domain (e.g., running a given IGP) to another (e.g., running a different IGP). Unfortunately, route redistribution may be responsible for both routing (inability to converge to a stable state) and forwarding (e.g., loop) anomalies [Le et al. 2008]. Generalized network update procedures have been proposed in [Vissicchio et al. 2014] to avoid transient anomalies while (i) reconfiguring a specific routing domain without impacting another, and/or (ii) arbitrarily changing the size and shape of routing domains.

Internet Service Providers (ISPs), with BGP and MPLS. In ISP networks, the Border Gateway Protocol (BGP) and often the MultiProtocol Label Switching (MPLS) protocol are pervasively used to manage transit traffic, for which both the source and the destination is external to the network. Vanbever et al. [2013b] showed that even techniques guaranteeing safe IGP reconfigurations can cause transient forwarding loops in those settings, because of the interaction between IGP and BGP. They also proved conditions to avoid those BGP-induced loops during IGP reconfigurations, by leveraging the presence of MPLS or carefully configuring BGP (according to some guidelines).

In parallel, a distinct set of techniques aimed at supporting BGP reconfigurations. Those contributions range from mechanisms to avoid churn of BGP messages during programmed operations (e.g., router reboots or BGP session maintenance [François et al. 2007a]) to techniques for safely moving virtual routers [Wang et al. 2008] or part of physical-router configuration (e.g., BGP sessions) [Keller et al. 2010]. A framework that guarantees strong consistency for arbitrary changes of the BGP configuration is presented in [Vissicchio et al. 2013]: It is based on implemeting Ships-in-the-Night in BGP and using packet tags to uniformly apply either the initial or the final forwarding at all routers.

Internet-level problems, like maintaining global connectivity upon failures, have also been explored (see, e.g., [Kushman et al. 2007]).

Protocol-independent reconfiguration frameworks. By design, all the above approaches are dependent on the considered (set of) protocols and even on their implementation.

Protocol-independent reconfiguration techniques have also been proposed. Mainly, in [2008], Alimi *et al.* generalize SITN by proposing a new design for the internal router architecture. This re-design would allow routers not only to run multiple configurations simultaneously, but also to select the configuration to be used for every packet based on the value of a specific bit in the packet header. The authors also describe a commit protocol to support the switch between configurations without creating forwarding loops.

General mechanisms for consensus routing have also been explored in [John et al. 2008].

2.3. Software-Defined Networks

Recently, Software Defined Networking has grown in popularity, thanks to its promises to spur abstractions, mitigate compelling management problems and avoid network ossification [McKeown et al. 2008]. Software-defined networks differ from traditional ones from an architectural viewpoint: In SDNs, the control is outsourced and consolidated to a logically-centralized element, the network controller, rather than having devices (switches and routers) run their own distributed control logic. In pure SDN networks, the controller computes (according to operators' input) and installs (on the controlled devices) the rules to be applied to packets traversing the network: No message exchange or distributed computation are needed anymore on network devices.

Fig. 6 depicts an example of an SDN network, configured to implement the initial state of our update example (see Fig. 1). Beyond the main architectural components, the figure also illustrate a classic interaction between them. Indeed, the dashed lines indicate that the SDN controller instructs the programmable network devices, typically switches [McKeown et al. 2008]), on how to process (e.g., forward) the traversing packets. An example command sent by the controller to switch s is reported next to the dashed line connecting the two: This command instructs s to use v_1 as next-hop for any packet destined to d.

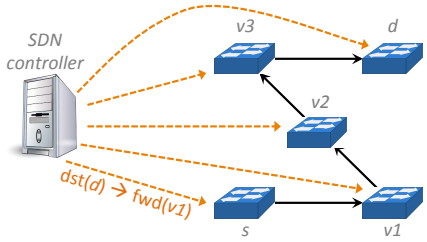
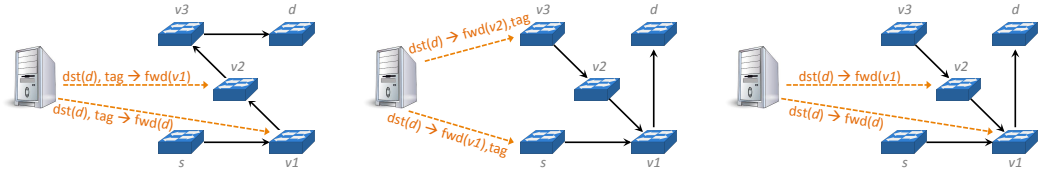


Fig. 6. Implementation of the surpassed state in Fig. 1 in an SDN network.

The SDN architecture is expected to make network updates more frequent and more critical than in traditional networks. On the one hand, controllers are often intended to support several different requirements, including performance (like optimal choice of per-flow paths), security (like firewall and proxy traversal) and packet-processing (e.g., through the optimized deployment of virtualized network functions) ones. On the other hand, devices cannot provide any reaction (e.g., to topological changes) like in traditional networks. In turn, this comes at the risk of triggering inconsistencies, e.g., creating traffic blackholes during an update, that are provably impossible to trigger by reconfiguring current routing protocols [Vissicchio et al. 2015]. As a consequence, the controller has to carry out a network update for every event (from failures to traffic surges and requirement modification) that can impact the forwarding rules installed on the switches; additionally, it should perform such updates while typically supporting more critical consistency guarantees (e.g., security-related ones) and performance objectives (e.g., for prompt reaction to failures) than in traditional networks.

An extended corpus of SDN update techniques have already been proposed in the literature, following up on the large interest raised by SDN in the last few years. This research effort nicely complements approaches to specify [Foster et al. 2013], compile [Casado et al. 2007; Monsanto

et al. 2013], and check the implementation of [Kazemian et al. 2013; Kazemian et al. 2012] network requirements that operators may want to implement in their (SDN) networks.



(a) Step 1 (b) Step 2 (c) Step 3

Fig. 7. Application of the 2-phase commit technique for carrying out our update example (see Fig. 3). A final (optional) step consists in cleaning the configuration by removing packet tags, i.e., reverting tagging at $v3$ and s as enforced by Step 2.

Historically speaking, the first cornerstone of SDN updates is represented by the work by Reitblatt et al. in [2011; 2012]. This work provides a first analysis of the additional (e.g., security) requirements to be considered for SDN updates, extending the scope of consistency properties from forwarding to policy ones. In particular, it focuses on per-packet consistency property, imposing that packets have to be forwarded either on their initial or on their final paths (never a combination of the two), throughout an update.

The technical proposal is centered around the 2-phase commit technique, which relies on tagging packets at the ingress so that either all initial rules or all final ones can be consistently applied network-wide. Initially, all packets are tagged with a given "old label" (e.g., no tag) and rules matching the old label are pre-installed on the switches. In a first step, the controller then instructs the internal switches to apply the final forwarding rules to packets carrying a "new label" – even if no packet carries such label at this step. After the internal switches have confirmed the successful installation of these new rules, the controller then changes the tagging policy at the ingress switches, requiring them to tag packets with the "new label". As a result, packets are immediately forwarded along the new paths. Finally, the internal switches are updated (to remove the old rules), and an optional cleaning step can be applied to remove all tags from packets. Fig. 7 shows the operational sequence produced by the 2-phase commit technique for the update case in Fig. 3.

Several works have been inspired by the 2-Phase technique presented in [Reitblatt et al. 2012]. On the one hand, a large set of contributions focused on additional guarantees that can be provided by building upon that technique, e.g., to avoid congestion during SDN updates (from [Hong et al. 2013] to [Brandt et al. 2016c; Brandt et al. 2016b; Förster and Wattenhofer 2016; Jin et al. 2014; Liu et al. 2013; Luo et al. 2015; Zheng et al. 2015]). On the other hand, several algorithms [Dudycz et al. 2016; Förster et al. 2016; Ludwig et al. 2016; Ludwig et al. 2015; Vanbever et al. 2012] to compute a set of ordered rule replacements have been proposed to deal with specific SDN update cases (e.g., where only forwarding consistency is needed) avoid adding rules and wasting critical network resources (i.e., expensive and precious switch TCAM memory slots).

In the following sections, we detail most of those contributions and the insights on different update problems that globally emerge from them.

3. TAXONOMY

We now present a general formulation of network update problem (§3.1), which abstracts from assumptions and settings considered in the literature. This formulation enables us to classify research contributions on the basis of the proposed techniques (e.g., simultaneous usage of multiple configurations on nodes or not) and algorithms, independently of their application to traditional and SDN networks (§3.2).

3.1. Generalized Network Update Problems

In order to compare and contrast research contributions, we first provide a generalized statement for network update problems. We use again Fig. 1 for illustration.

Basic Problem. Generally speaking, a network update problem consists in computing a sequence of operations that changes the packet-processing rules installed on network devices. Consider any communication network: It is composed by a given set of inter-connected devices, that are able to process data packets (e.g., forwarding them to a next-hop) according to rules installed on them. We refer to the set of rules installed on all devices at a given time as network state at that time. Given an initial and final state, a network update consists in passing from the initial state to the final one by applying operations (i.e., adding, removing or changing rules) on different devices. In Fig. 1, the initial state forces packets from source s to destination d along the path (s, v_1, v_2, v_3, d); the final state forwards the same packets over (s, v_1, d), and packets from v_3 to d on (v_3, v_2, v_1, d). The network update problem consists in replacing the initial rules with the final ones, so that the paths for d are updated from (s, v_1, v_2, v_3, d) to (s, v_1, d) and (v_3, v_2, v_1, d).

Operations. To perform a network update, a sequence of operations has to be computed. By operation, we mean a (direct or indirect) modification of packet-processing rules installed on one or more device. As an example, an intuitive and largely-supported operation on network devices is rule replacement, which consists in instructing a device (e.g., v_3) to replace an initial rule (e.g., forward the $s - d$ packet flow to v_2) with the corresponding final one (e.g., forward the $s - d$ flow to d). Operations can be coarse-grained and indirect, as IGP link reweighting or configuration swapping in legacy networks that imply multiple rule replacements at distinct devices (see §2).

Consistency. The difficulty in solving network update problems is that some form of consistency must be guaranteedly preserved during the update, for practical purposes (e.g., avoiding service disruptions and packet losses). Preserving consistency properties, in turn, depends on the order in which operations appear in the computed sequence and are executed by network devices. For example, if v_3 replaces its initial rule with its final one before v_2 in Fig. 1, then the operational sequence triggers a forwarding loop between v_2 and v_3 that interrupts the connectivity from s to d. In §3.2, we provide an overview of consistency properties considered in the literature.

The practical need for guaranteeing consistency has two main consequences (as shown in §2). First, it forces network updates to be performed incrementally, i.e., appropriately scheduling operations over time so that the installed intermediate states are provably disruption-free. Second, it requires a careful computation of operational sequences, implementing specific reasoning in the problem-solving algorithms (e.g., to avoid replacing v_3's rule before v_2's one in the previous example).

Performance. Another algorithmic challenge consists in optimizing network-update performance. As an example, minimizing the time to complete an update is commonly considered among those optimization objectives. Indeed, carrying out an update incrementally requires to install intermediate configurations, and in many cases it is practically desirable to minimize the time spent in such intermediate states. We provide a broader overview of performance goals considered by previous works in §3.2.

Final Operational Sequences. Generally, the solution for an update problem can be represented as a sequence of *steps* or *rounds*, that both (i) guarantees consistency properties and (ii) optimizes update performance. Each step is a set of operations that can be started at the same time. Note that this does not mean that operations in the same step are assumed to be executed simultaneously on the respective devices; Rather, all operations in the same step can be started in parallel because the target consistency properties are guaranteed independently of the relative order in which those operations are executed. Examples of operational sequences, computed by different techniques, are reported in §2 (see Figs. 3 and 5).

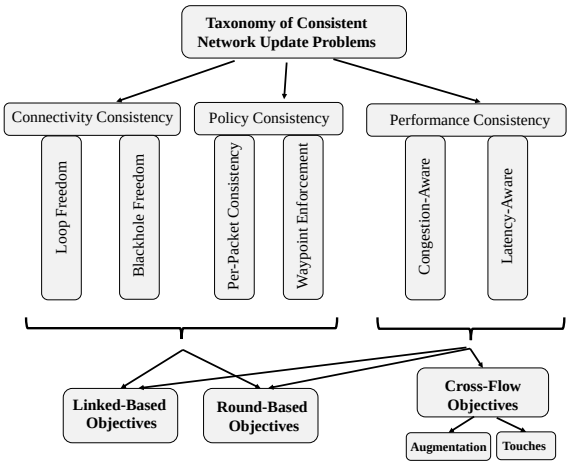


Fig. 8. Types of consistent network update problems, defined independently of the network setting (rule granularity and supported operations).

3.2. Update Techniques

In this section, we provide an overview of the problem space and classify existing models and techniques. Previous contributions have indeed considered several variants of the generalized network update problem as we formulated in §3.1. Those variants basically differ in terms of the update problem on which they focus. Update problems in turn define both (1) the network setting, including admitted rule granularity and operations, (2) consistency properties, and (3) performance goals. An overview of the main update problems considered by previous works is visualized in Fig. 8, where we skipped the orthogonal network setting dimensions for clarity.

Rule Granularity. Network update techniques assume that the underlying devices support rules that implement one of the two alternative routing models: *destination-based* and *per-flow* routing.

(1) **Destination-based Routing:** In destination-based routing, routers forward packets based on the destination only. An example for destination-based routing is IP routing, where routers forward packets based on the longest common IP destination prefix. In particular, destination-based routing describes confluent paths: once two flows from different sources destined toward the same destination intersect at a certain node, the remainder (suffix) of their paths will be the same. In destination-based routing, routers store at most one forwarding rule per specific destination.
(2) **Per-flow Routing:** In contrast, according to *per-flow* routing, routes are not necessarily confluent: the forwarding rules at the routers are defined per-flow, i.e., they may depend not only on the destination but for example also on the source. In traditional networks, flows and per-flow routing could for example be implemented using MPLS: packets belonging to the same equivalence class resp. packets with the same MPLS tag are forwarded along the same path.

Operations. Techniques to carry out network updates can be classified in broad categories, depending on the operations that they consider.

(1) **Rule replacements:** A first class of update techniques is based on computing an order in which initial rules are replaced by the corresponding final ones on the devices. Depending on the target setting (e.g., legacy networks or SDNs), such replacement can be admitted at different granularity, that is, on a per-rule and per-device basis (as in OpenFlow networks) or on a per-group of rules and devices (as for link reweighting in IGP).

(2) **Rule additions:** A second class of network update algorithms is based on adding rules to guarantee consistency during the update. The following two main variants of this approach have been explored so far.
   (a) **2-Phase commit:** In this case, both the initial and the final rules are installed on all devices in the central steps of the updates. Packets are tagged at the border of the network to enforce that the internal devices either (i) all use the initial rules, or (ii) all use the final rules. See Fig. 7 for an example.
   (b) **Helper rules:** Some techniques introduce additional rules, which do not belong neither to the old state nor to the new one, in some intermediate update step. These rules allow to divert the traffic temporarily to other parts of the network, and are called *helper rules*.
(3) **Mixed:** Recently, some update techniques combine rule replacements and additions, in order to reduce the update overhead (especially in terms of device-memory consumption) while keeping the flexibility provided by adding rules.

Consistency properties. Update techniques typically target to preserve one (or more) of the following consistency properties. This dimension is also reflected in the structure of this survey.

(1) **Connectivity consistency:** The most basic form of consistency regards the capability of the network to keep delivering packets to their respective destinations, throughout the update process. This boils down to guaranteeing two correctness properties: absence of blackholes (i.e., paths including routers that cannot forward the packets further) and absence of forwarding loops (i.e., packets bouncing back and forth on a limited set of routers, without reaching their destinations).
(2) **Policy consistency:** Paths used to forward packets may be selected according to specific forwarding policies, for example, security ones imposing that given traffic flows must traverse specific waypoints (firewalls, proxies, etc.). In many cases, those policies have to be preserved during the update. Generally speaking, policy consistency properties impose constraints on which paths can be installed during the update. For example, an already-mentioned policy consistency property (see §2) is *per-packet consistency*, requiring that packets are always forwarded along either the pre-update or the post-update paths, but never a combination of the two.
(3) **Performance consistency:** A third class of consistency properties takes into account the actual availability and limits of network resources. For instance, many techniques account for traffic volumes and corresponding constraints raised by the limited capacity of network links: Those techniques aim at respecting link-capacity constraints in each update step, e.g., to avoid *transient congestion* during updates.

Performance goals. We can distinguish between three broad classes of performance goals.

(1) **Link-based:** A first class of consistent network update protocols aims to make new links available as soon as possible, i.e., to maximize the number of switch rules which can be updated simultaneously without violating consistency.
(2) **Round-based:** A second class of consistent network update protocols aims to minimize the number of inter-actions between the controller and the switches.
(3) **Cross-Flow Objectives:** A third class of consistent network update protocols targets objectives arising in the presence of multiple flows.
   (a) **Augmentation:** Minimize the extent to which link capacities are oversubscribed during the update (or make the update entirely congestion-free).
   (b) **Touches:** Minimize the number of interactions with the router.

Link-based and round-based objectives are usually considered for node-ordering algorithms and for weak-consistency models. Congestion-based objectives are naturally considered for capacitated consistency models.

4. CONNECTIVITY CONSISTENCY

In this section, we focus on update problems where the main consistency property to be guaranteed concerns the delivery of packets to their respective destinations. Packet delivery can be disrupted during an update by forwarding loops or blackholes transiently present in intermediate states. We separately discuss previous results on how to guarantee loop-free and blackhole-free network updates. We start from the problem of avoiding forwarding loops during updates, because they are historically the first update problems considered – by works on traditional networks (see §2). This is also motivated by the fact that blackholes cannot be created by reconfiguring current routing protocols, as proved in [Vissicchio et al. 2015]. We then shift our focus on avoiding blackholes during arbitrary (e.g., SDN) updates.

4.1. Loop-Freedom

Loop-freedom is a most basic consistency property and has hence been explored intensively already.

4.1.1. Definitions. We distinguish between flow-based and destination-based routing: in the former, we can focus on a single (and arbitrary) path from s to d: forwarding rules stored in the switches depend on both s and d, can flows can be considered independently. In the latter, switches store a single forwarding rule for a given destination: once the paths of two different sources destined to the same destination intersect, they will be forwarded along the same nodes in the rest of their route: the routes are confluent.

Moreover, once can distinguish between two different definitions for loop-free network updates: *Strong Loop-Freedom (SLF)* and *Relaxed Loop-Freedom (RLF)* [Ludwig et al. 2015]. SLF requires that at any point in time, the forwarding rules stored at the switches should be loop-free. RLF only requires that forwarding rules stored by switches *along the path from a source s to a destination d* are loop-free: only a small number of "old packets" may temporarily be forwarded along loops.

4.1.2. Algorithms and Complexity. **Node-based objective ("greedy approach").** Mahajan and Wattenhofer [2013] initiated the study of destination-based (strong) loop-free network updates. In particular, the authors show that by scheduling updates across multiple rounds, consistent update schedules can be derived which do not require any packet tagging, and which allow some updated links to become available earlier. The authors also present a first algorithm that quickly updates routes in a transiently loop-free manner. The study of this model has been refined in [Förster et al. 2016; Förster and Wattenhofer 2016], where the authors also establish hardness results. In particular, the authors prove that for two destinations and for sublinear x, it is NP-hard to decide if x rounds (cf. round-based objectives) of updates suffice. Furthermore, maximizing the number of rules updated for a single destination is NP-hard as well, but can be approximated well.

Ludwig et al. in [Ludwig et al. 2015] and [Ludwig et al. 2014] initiated the study of arbitrary route updates: routes which are not necessarily destination-based. The authors show that the update problem in this case boils down to an optimization problem on a very simple directed graph: initially, before the first update round, the graph simply consists of two connected paths, the old and the new route. In particular, every network node which is not part of both routes can be updated trivially, and hence, there are only three types of nodes in this graph: the source s has out-degree 2 (and in-degree 0), the destination d has in-degree 2 (and out-degree 0), and every other node has in-degree and out-degree 2. The authors also observe that loop-freedom can come in two flavors, strong and relaxed loop-freedom [Ludwig et al. 2015].

Despite the simple underlying graph, however, Amiri et al. [2016] show that the node-based optimization problem is NP-hard, both in the strong and the relaxed loop-free model (SLF and RLF). As selecting a maximum number of nodes to be updated in a given round (i.e., the node-based optimization objective) may also be seen as a heuristic for optimizing the number of update rounds (i.e., the round-based optimization objective), the authors refer to the node-based approach as the "greedy approach".

Amiri et al. [2016] also present polynomial-time optimal algorithms for the following scenarios: Both a maximum SLF update set as well as a maximum RLF update set can be computed in polynomial-time in trees with two leaves. Regarding polynomial-time approximation results, the problem is 1/2-approximable in general, both for strong and relaxed loop-freedom. For additional approximation results for specific problem instances, we refer the reader to Amiri et al. [2016].[1]

Round-based objective ("greedy approach"). Ludwig et al. [2015] initiate the study of consistent network update schedules which minimize the number of interaction rounds with the controller: *How many communication rounds k are needed to update a network in a (transiently) loop-free manner?*

The authors show that answering this question is difficult in the strong loop-free case. In particular, they show that while deciding whether a k-round schedule exists is trivial for $k = 2$, it is already NP-complete for $k = 3$. Moreover, the authors show that there exist problem instances which require $\Omega(n)$ rounds, where n is the network size. Moreover, the authors show that the greedy approach, aiming to "greedily" update a *maximum* number of nodes in each round, may result in $\Omega(n)$-round schedules in instances which actually can be solved in $O(1)$ rounds; even worse, a *single* greedy round may inherently delay the schedule by a factor of $\Omega(n)$ more rounds.

However, fast schedules exist for *relaxed loop-freedom*: the authors present a deterministic update scheduling algorithm which completes in $O(\log n)$-round in the worst case.

Hybrid Approaches. Vissicchio and Cittadini [2016] presented FLIP, which combines per-packet consistent updates with order-based rule replacements, in order to reduce memory overhead: additional rules are used only when necessary. Moreover, Hua et al. [2016] initiated the study of adversarial settings, and presented FOUM, a flow-ordered update mechanism that is robust to packet-tampering and packet dropping attacks.

Other Objectives. Dudycz et al. [2016] initiated the study of how to update multiple policies simultaneously, in a loop-free manner. In their approach, the authors aim to minimize the number of so-called *touches*, the number of updates sent from the controller to the switches: ideally, all the updates to be performed due the different policies can be sent to the switch in one message. The authors establish connections to the *Shortest Common Supersequence (SCS)* and *Supersequence Run* problems [Middendorf 1994], and show NP-hardness already for two policies, each of which can be updated in two rounds, by a reduction from *Max-2SAT* [Lewin et al. 2002].

However, the authors also present optimal polynomial-time algorithms to combine consistent update schedules computed for individual policies (e.g., using any existing algorithm, e.g., [Ludwig et al. 2015; Mahajan and Wattenhofer 2013]), into a global schedule guaranteeing a minimal number of touches. This optimal merging algorithm is not limited to loop-free updates, but applies to any consistency property: if the consistency property holds for individual policies, then it also holds in the joint schedule minimizing the number of touches. the *Shortest Common Supersequence (SCS)* and *Supersequence Run* [Middendorf 1994].

4.1.3. Related Optimization Problems. The link-based optimization problem, the problem of maximizing the number of links (or equivalently nodes) which can be updated simultaneously, is an instance of the maximum acyclic subgraph problem; equivalently, the dual problem of minimizing the number of links which cannot be updated is a minimum feedback arc set problem. For showing NP-hardness, reductions from the *minimum hitting set problem* [Amiri et al. 2016] uses the *feedback arc set problem* [Förster and Wattenhofer 2016] are shown. A more abstract reconfiguration framework on how to transform a feasible solution of a problem into another solution of the same problem is also studied from a purely theoretical point of view (e.g., [Kaminski et al. 2011]).

[1] We note that there exists a subtle difference between the approximation results by Foerster et al. and Ludwig et al.: the former authors usually aim to minimize the number of links which *cannot* be updated [Förster and Wattenhofer 2016] (a feedback arc set problem), while Ludwig et al. [Amiri et al. 2016] consider the dual problem variant and aim to maximize the links which *can* be updated in the given round (the maximum acyclic subgraph problem). The approximation guarantees of the two problems differ: for the former model, the best known approximation bound is $O(\log n \log \log n)$ [Even et al. 1998] while for the latter, constant approximation results exist.

4.1.4. Concluding Remarks and Open Problems. Loop-free network updates still pose several open problems. Regarding the node-based objective, Amiri et al. [2016] conjecture that update problems on bounded directed path-width graphs may still be solvable efficiently: none of the negative results for bounded degree graphs on graphs of bounded directed treewidth seem to be extendable to digraphs of bounded directed pathwidth with bounded degree. More generally, the question of on which graph families network update problems can be solved optimally in polynomial time in the node-based objective remains open. Regarding the round-based objective, it remains an open question whether strong loop-free updates are NP-hard for any $k \geq 3$ (but smaller than n): so far only $k = 3$ has been proved to be NP-hard. More interestingly, it remains an open question whether the relaxed loop-free update problem is NP-hard, e.g., are 3-round update schedules NP-hard to compute also in the relaxed loop-free scenario? Moreover, it is not known whether $\Omega(\log n)$ update rounds are really needed in the worst-case in the relaxed model, or whether the problem can always be solved in $O(1)$ rounds. Some brute-force computational results presented in [Ludwig et al. 2015; Mahajan and Wattenhofer 2013] indicate that if it is constant, the constant must be large.

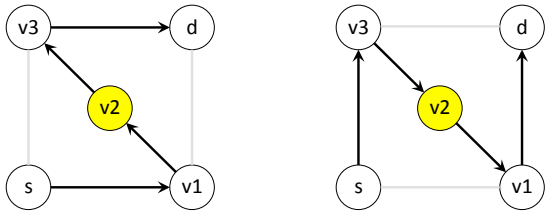
Table I. Overview of results for loop-freedom.

Model	NP-hard	Polynomial time	Remarks
# Rounds, strong LF	Is there a 3-round loop-free update schedule? [Ludwig et al. 2015] *For 2-destination rules and sublinear x: Is there a x-round loop-free update schedule? [Förster and Wattenhofer 2016]*	Is there a 2-round loop-free update schedule? [Ludwig et al. 2015]	In the worst case, $\Omega(n)$ rounds may be required. [Ludwig et al. 2015], *[Förster et al. 2016]*. $O(n)$-round schedules always exist *[Mahajan and Wattenhofer 2013]*. Both applies to flow-based & *destination-based* rules.
# Rounds, relaxed LF	No results known.	$O(\log n)$-round update schedules always exist. [Ludwig et al. 2015]	It is not known whether $o(\log n)$-round schedules exist (in the worst case). No approximation algorithms are known.
# Links, strong LF	Is it possible to update x nodes in a loop-free manner? [Amiri et al. 2016], *[Förster and Wattenhofer 2016]*	Polynomial-time optimal algorithms are known to exist in the following cases: A maximum SLF update set can be computed in polynomial-time in trees with two leaves. [Amiri et al. 2016]	The optimal SLF schedule is 2/3-approximable in polynomial time in scenarios with exactly three leaves. For scenarios with four leaves, there exists a polynomial-time 7/12-approximation algorithm. [Amiri et al. 2016] Approximation algorithms from maximum acyclic subgraph [Amiri et al. 2016] and minimum feedback arc set *[Förster et al. 2016]* apply.
# Links, relaxed LF	Is it possible to update x nodes in a loop-free manner? [Amiri et al. 2016]	Polynomial-time optimal algorithms are known to exist in the following cases: A maximum RLF update set can be computed in polynomial-time in trees with two leaves. [Amiri et al. 2016]	No approximation results known. [Amiri et al. 2016]

Note: Results/references in *italics* are in the destination-based model.

4.2. Blackhole-Freedom

Another consistency property is blackhole freedom, i.e., a switch should always have a matching rule for any incoming packet, even when rules are updated (e.g., removed and replaced). This property is easy to guarantee by implementing some default matching rule which is never updated, which however could in turn induce forwarding loops. A straightforward mechanism, if there is currently no blackhole for any destination, is to install new rules with a higher priority, and then delete the old rules [Förster et al. 2016; Mahajan and Wattenhofer 2013]. Nonetheless, in the presence of memory limits and guaranteeing loop-freedom, finding the fastest blackhole-free update schedule is NP-hard [Förster et al. 2016].



(a) Surpassed state (b) New state

Fig. 9. A WPE-consistent update example, taken from [Ludwig et al. 2014], where forwarding paths have to be changed from the Surpassed (Fig. 9a) to the New (Fig. 9b) state, while preserving traversal of the waypoint $v2$ (highlighted in the figure) at any time during the update.

5. POLICY CONSISTENCY

While connectivity invariants are arguably the most intensively studied consistency properties in the literature, especially in traditional networks, operators often have additional requirements to be preserved. For example, operators want to ensure that packets traverse a given middlebox (e.g., a firewall) for security reasons or a chain of middleboxes (e.g., encoder and decoder) for performance reasons, or that paths comply with Service Level Agreements (e.g., in terms of guaranteed delay). In this section, we discuss studied problems and proposed techniques aiming at preserving such additional requirements.

5.1. Definitions

Additional requirements on forwarding paths that may have to be respected during a network update can be modeled by *routing policies*, that is, sub-paths that have to be traversed by transient paths installed during network updates.

Over the years, several contributions have targeted policy-preserving updates, typically focusing on specific policies. Historically, the first policy considered during network updates is *per-packet consistency (PPC)*, which ensures that every packet travels either on its initial or on its final paths, never on intermediate ones. This property is the most natural to (try to) preserve. Assume indeed that both the initial and the final paths comply with high-level network requirements, e.g., security, performance, SLA policies. The most straightforward way to guarantee that those requirements are not violated is to constrain all paths installed during the update to always be either initial paths or final ones.

Nonetheless, guaranteeing per-packet consistency may be an unnecessarily strong requirement in practice. Not always it is strictly needed that transient paths must coincide with either the initial or the final ones. For example, in some cases (e.g., for enterprise networks), security may be a major concern, and many security requirements may be enforced by guaranteeing that packets traverse a firewall. We refer to this specific case where single nodes (waypoints) have to be traversed by given traffic flows as *waypoint enforcement (WPE)*. An example WPE-consistent update is displayed in Fig. 9

More complex policies (i.e., beyond WPE) may also be needed in general. Indeed, policies to be satisfied in SDN networks tend to grow in number and complexity over time, because of both new requirements (e.g., as dictated by use cases like virtualized infrastructure and network functions) and novel opportunities (e.g., programmability and flexibility) opened by SDN. For example, it may desirable that specific traffic flows follow certain sub-paths (e.g., with low delay for video streaming and online gaming applications) or are explitictly denied to pass through other sub-paths (e.g., because of political or economical constraints). Such *arbitrary policies* are also considered in recent SDN update works.

5.2. Algorithms and Complexity

2-Phase commit techniques. As described in §3.2, 2-Phase commit techniques deploy carry out updates by (1) tagging packets at their ingress in the network, and (2) using packet tags to use initial or final paths consistently network-wide. Unsurprisingly, this approach guarantees per-packet consistency (hence, potentially any policy satisfied by both pre- and post-update paths).

While the idea is quite intuitive, some support is needed on the devices, e.g., to tag packets and match packet tags. A framework to implement this update approach in traditional networks has been proposed by Alimi et al. [2008]. It requires invasive modification of router internals, to manage tags and run arbitrary routing processes in separate process spaces. The counterpart of such a framework for SDN networks is presented in [Reitblatt et al. 2012; Reitblatt et al. 2011]. Those works avoid the need for changing device internals since it relies on OpenFlow, the protocol classically used in SDN networks. They also argue on the criticality of supporting PPC in the SDN case and the advantages of integrating 2-phase commit techniques within an SDN controller.

A major downside of 2-phase commit is that it doubles the consumed memory on switches, along with requiring header space, tagging overhead, and complications with middleboxes changing tags [Qazi et al. 2013], [Fayazbakhsh et al. 2013]. It indeed 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 mitigate this problem, a variant of the basic approach has been studied in [Katta et al. 2013]. The authors of the latter work proposed to break a given update into several sub-updates, such that each sub-update changes the paths for a different set of flows. Of course, this approach would make it longer for the full update to be completed. 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.

Actually, the switch-memory consumption of 2-phase commit techniques remains a fundamental limitation of the approach, which also motivated the exploration of alternatives.

SDN-based update protocols. McGeer [2012; 2013] presented two protocols to carry out network updates and defined on top of OpenFlow. The first update protocol [McGeer 2012] saves switch resources by sending packets to the controller during updates. As a result, switch resources (like precious TCAM entries) are saved, at the cost of adding delay on packet delivery, and consuming network bandwidth and controller memory. The second update protocol [McGeer 2013] is based on a logic circuit for the update sequence which requires neither rule-space overhead nor transferring the packets to the shelter during the update.

Both proposals need a dedicated protocol currently supported by devices out of the box.

Rule replacement ordering. Some works explored which policies can be supported, and how, by only relying on (ordered) rule replacements, given that this both (i) comes with no memory overhead and (ii) is supported by both traditional and SDN devices.

Some works noticed that PPC can be an unnecessarily strong requirement in several practical cases. Initial contributions mainly focused on WPE consistency, e.g., to preserve security policies. Prominently, [Ludwig et al. 2014] studies how to compute quick updates that preserve WPE by only replacing initial with final rules, when any given flow has to traverse a single waypoint. The authors propose WayUp, an algorithm that guarantees WPE during the update and terminates in 4 rounds. However, they also show that it may not be possible to ensure waypointing through a single node and loop-freedom at the same time. Fig. 9 actually shows one case in which any rule replacement ordering either causes a loop or a WPE consistency violation. Those infeasibility results are extended to waypoint chains in [Ludwig et al. 2016]. In that work, in particular, the authors show that flexibility in ordering and placing virtualized functions specified by a chain do not make the update problem always solvable. The two works also show that it is computationally hard (NP-hard) to even decide if an ordering preserving both WPE and loop-freedom exists. Mixed integer program formulations to find an operational are proposed and evaluated in both cases.

The more general problem of preserving policies defined by operators is tackled in [McClurg et al. 2015]. That paper describes an approach to (i) model update-consistency properties as Linear

Temporal Logical formulas, and (ii) automatically synthesize SDN updates that preserve input properties. Such a synthesis is performed by an efficient algorithm based on counterexample-guided search and incremental model checking. Experimental evidence is provided about the scalability of the algorithm (up to one-thousand node networks).

Finally, [Vissicchio et al. 2013] explores algorithmic limitations of guaranteeing per-packet consistency without relying on state duplication. The work shows that a greedy strategy implements a correct and complete approach in this case, meaning that it finds the maximal sequence of rule replacements that do not violate PPC. Cerný et al. [2016] complement those findings, by presenting a polynomial-time synthesis algorithm that preserves PPC while allowing the maximal parallelism between per-switch updates. Also, an evaluation on realistic update cases is presented in [Vissicchio et al. 2013]. It shows that PPC can be preserved while replacing many forwarding entries on the majority of the switches, despite updates can rarely be completed this way. However, this observation motivates both approaches tailored to a more restricted family of policies (like WPE-preserving ones, described above), and efforts for mixed approaches (mixing rule replacements and duplication, see below).

Mixed approaches. In [Vissicchio et al. 2013], a basic mixed approach is considered to ensure PPC in generalized networks running both traditional and SDN control-planes (or any of the two). This approach consists in first computing the maximal sequence of rule replacements that preserve PPC, and then applying a restricted 2-phase commit procedure on a subset of (non-ordered) devices and flows.

Vissicchio and Cittadini [2016] propose an algorithm addressing a larger set of update problems with a more general algorithmic approach, but restricting to SDN networks. This work focuses on the problem of preserving generic policies during SDN updates. For each flow, a policy is indeed defined as a set of paths so that the flow must traverse any of those paths in each intermediate state. The proposed algorithm interleaves 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 primitives in subsequent steps as in [Vissicchio et al. 2013].

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

5.3. Related Optimization Problems

Many policy-preserving algorithms face generalized versions of the optimization problems associated to connectivity-preserving updates (see §4): While the most common objective remains the maximization of parallel operations (to speed-up the update), policy consistency requires that all possible intermediate paths comply with certain regular expressions in addition to being simple (that is, loop-free) paths. Mixed policy-preserving approaches focus on even more general problems where (i) different operations can be interleaved in the output operational sequence (which provides more degrees of freedom in solving the input problems), and (ii) multiple optimization objectives are considered at the same time (typically, maximizing the update parallelism while also minimizing the consumed switch memory).

5.4. Concluding Remarks and Open Problems

Unsurprisingly, preserving policies requires more sophisticated update techniques, since it is generally harder to extract policy-induced constraints and model the search space. Two major families of solutions have been explored so far. On the one hand, 2-phase commit techniques and update protocols sidestep the algorithmic challenges, at the cost of relying on specific primitives (packet tagging and tag matching) that comes with switch memory consumption. On the other hand, ordering-based techniques directly deal with problem complexities, at the cost of algorithmic simplicity and impossibility to always solve update problems. Finding the best balance between those two extremes is an interesting research direction. Some initial work has started in this direction, with the proposal of algorithms that can interleave different kinds of operations within the computed

sequence (see mixed approaches in §5.2). However, many research questions are left open. For example, the computational complexity of solving update problems while mixing rule additions (for packet tagging and matching) with replacements is unknown. Moreover, it is unclear whether the proposed algorithms can be improved exploiting the structure of specific topologies or the flexibility of new devices (e.g., P4-compatible ones [Bosshart et al. 2014]), e.g., to achieve better trade-offs between memory consumption and update speed.

6. PERFORMANCE-AWARE CONSISTENCY

Computer networks are inherently capacitated, and respecting resource constraints is hence another important aspect of consistent network updates. Congestion is known to significantly impact throughput and increase latency, therefore negatively impacting user experience and even leading to unpredictable economic loss.

6.1. Definitions

The capacitated update problem is to migrate from a multi-commodity flow $\mathcal{F}_{old}$ to another multi-commodity flow $\mathcal{F}_{new}$, where consistency is defined as not violating any link capacities and not rate-limiting any flow below its demand in $\min(\mathcal{F}_{old}, \mathcal{F}_{new})$. In few works, e.g., [Brandt et al. 2016c], $\mathcal{F}_{new}$ is only implicitly specified by its demands, but not by the actual flow paths. Some migration algorithms will violate consistency properties to guarantee completion, as a consistent migration does not have to exist in all cases.

Typically, four different variants are studied in the literature: First, individual flows may either only take one path (unsplittable) or they may follow classical flow-theory, where the incoming flow at a switch must equal its outgoing flow (splittable). Secondly, flows can take any paths via helper rules in the network during the migration (intermediate paths), or may only be routed along the old or the new paths (no intermediate paths).

To exactly pinpoint congestion-freedom, one would need to take many detailed properties into account, e.g., buffer sizes and ASIC computation times. As such, the standard consistency model does not take this fine-grained approach, but rather aims at avoiding ongoing bandwidth violations and takes a mathematical flow-theory point of view. Introduced by [Hong et al. 2013], consistent flow migration is captured in the following model: No matter if a flow is using the rules before the update or after the update, the sum of all flow sizes must be at most the links capacity.

6.2. Algorithms and Complexity

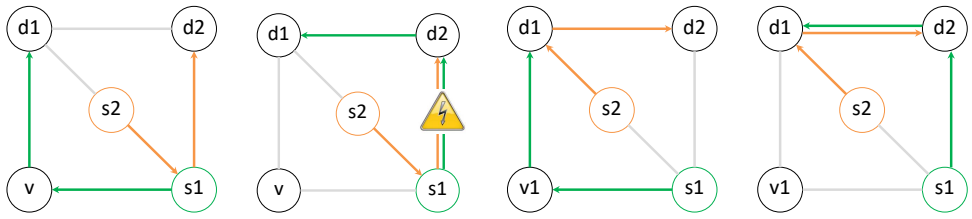
6.2.1. Algorithms. Current algorithms for capacitated updates of network flows use the seminal work by Reitblatt et al. [2012] as an update mechanism. Analogously to *per-packet consistency* (cf. §5), one can achieve *per-flow consistency* by a 2-phase commit protocol. While this technique avoids many congestion problems, is not sufficient for bandwidth guarantees: When updating the two flows in Fig. 10, the lower green flow could move up before orange flow is on its new path, leading to congestion.

An overview over all algorithmic approaches discussed here can be found in Table II.

Mizrahi and Moses [2014a] prove that flow swapping is necessary for throughput optimization in the general case, as thus algorithms are needed that do not violate any capacity contraints during the network update, beyond simple flow swapping as well.

The seminal work by Hong et al. [2013] on *SWAN* introduces the current standard model for capacitated updates. Their algorithmic contribution is two-fold, and also forms the basis for *zUpdate* [Liu et al. 2013]: First, the authors show that if all flow links have free capacity *slack* s, consistent migration is possible using $\lceil 1/s \rceil - 1$ updates: E.g., if the free capacity is 10%, 9 updates are required, always moving 10% of the links' capacity to the new flow paths. If the network contains non-critical background traffic, free capacity can be generated for a migration by rate-limiting this background traffic temporarily, cf. Fig. 11: removing some background traffic allows for consistent migration.

Second, the authors provide an LP-formulation for splittable flows which provides a consistent migration schedule with x updates, if one exists. By performing a binary search over the number of



(a) Initial (b) Congestion! (c) Orange first (d) Final

Fig. 10. In this introductory network for flow migration, all links have unit bidirectional capacity, and both orange and green flows have unit size as well. The task is to move both the green and orange flows from their initial paths in Fig. 10a to their final ones shown in Fig. 10d. Updating both flows together could lead to the green flow being moved first, inducing congestion, see Fig. 10b. However, this can be avoided by using succinct updates, first moving the orange flow as in Fig. 10c, and then moving the green flow.

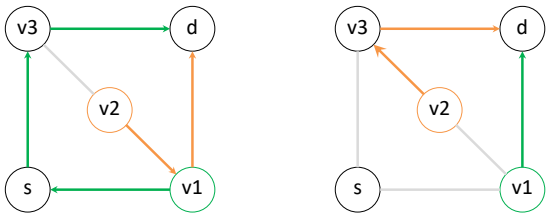
Table II. Compact overview of flow migration algorithms.

Ref.	Approach	(Un-)splittable model	Interm. paths	Computation	# Updates	Complete (decides if cons. migr. exists)
[Reitblatt et al. 2012]	Install old and new rules, then switch from old to new	Both, move each flow only once	No	Polynomial	1	No bandwidth guarantees
[Hong et al. 2013]	Partial moves according to free slack capacity s	Splittable	No	Polynomial	$\lceil 1/s \rceil - 1$	Requires slack on flow links
[Jin et al. 2014]	Greedy traversal of dependency graph	Both, move each flow only once	No	Polynomial	Linear	No (rate-limit flows to guarantee completion)
[Luo et al. 2015]	MIP of [Jin et al. 2014]	Both, move each flow only once	No	Exponential	Linear	Yes
[Zheng et al. 2015]	Minimize trans. congestion for fixed number of x intermediate states via	Both	No	Polynomial	Any $x \in \mathbb{N}$	For given x, approx. min. trans. congestion (if > 0) by $\log n$ factor
			Yes	*Exponential*		
[Zheng et al. 2015]	LPvia MIP	Both	Both	*Exponential*	Any $x \in \mathbb{N}$	For any given x yes, but not in general
[Hong et al. 2013]	Binary search of intermediate states via LP	Splittable	Yes	Polynomial in # of updates	Unbounded	Cannot decide if cons. migration possible
[Brandt et al. 2016b]	Create slack with intermediate states, then use partial moves of [Hong et al. 2013]	Splittable	Yes	Polynomial	Unbounded	Yes
[Förster and Wattenhofer 2016]	Split unsplittable flows along old and new paths	2-Splittable	No	Polynomial	Unbounded	Yes
[Brandt et al. 2016c]	Use augmenting flows to find updates	Splittable, 1 dest., paths not fixed	Yes	Polynomial	Linear	Yes

Further practical extensions	
[Liu et al. 2013]	Extends approach of *SWAN* [Hong et al. 2013] in a data center setting
[Wang et al. 2016]	Extends approach of *Dionysus* [Jin et al. 2014] with local dependency resolving
[Paris et al. 2016]	Considers reconfiguration for dynamic flow arrivals
[Luo et al. 2016]	Allows (un-)splittable flow migration (move once) with user-spec. deadlines & requirements via MIP (LP heuristic)

updates, the number of necessary updates can be minimized. This approach allows for intermediate paths, where the flows can be re-routed anywhere in the network. E.g., consider the example in Fig. 11 with all flows and links having unit size. If there was an additional third route to d, the orange flow could temporarily use this intermediate path: we can then switch the green flow, and eventually the orange flow could be moved to its desired new path.

This second LP-formulation was extended by Zheng et al. [2015] to include unsplittable flows as well via a MIP. Furthermore, using randomized rounding with an LP, Zheng et al. [2015] can approximate the minimum congestion that will occur if the migration has to be performed using x updates. Should intermediate paths be allowed however, then their LP is of exponential size. Paris et al. [2016] also consider the tradeoff between reconfiguration effects and update speed in the context



(a) Initial (b) Final

Fig. 11. In this network the task is again to migrate consistently from the initial to the final state. If all flows and links have unit size, no consistent migration is possible: The destination has just two incoming links of combined size two. Should the flows just have a size of $2/3$, one can migrate consistently in $\lceil 1/(1/3) \rceil - 1 = 2$ updates by moving half of the flow size of $1/3$ each time in parallel.

of dynamic flow arrivals. In terms of tradeoffs, Luo et al. [2016] allow for user-specified deadlines (e.g., a flow has to be updated until some time t) via an MIP or an LP-based heuristic.

The work by Brandt et al. [2016b] tackles the problem of deciding in polynomial time if consistent migration is possible at all for splittable flows with intermediate paths allowed. By iteratively checking for augmenting flows that create free capacity (slack) on fully-capacitated links, it is possible to decide in polynomial time if slack can be obtained on all flow links. If yes, then the first technique of [Hong et al. 2013] can be used, else no consistent migration is possible. Should the output be no, they also provide an LP-formulation to check to which demands it is possible to migrate consistently.

Jin et al. [2014] also consider the variable update times of switches in the network. For both splittable and unsplittable flows without intermediate paths, they build a dependency graph for the update problem. Then, this dependency graph is traversed in a greedy fashion, updating whatever flows are currently possible. E.g., in Fig. 10, the orange flow would be moved first, then the green flow next. Should this traversal result in a deadlock, flows are rate-limited to guarantee progress. Wang et al. [2016] improve the local dependency resolving to improve the greedy traversal, and Jin et al. [2016] furthermore consider circuit nodes. Luo et al. [2015] provide a MIP-formulation of the problem, and also provide a heuristic framework using tiny MIPs.

Förster and Wattenhofer [2016] consider an alternative approach to migrating unsplittable flows without intermediate paths: They split each flow along its old and new path, changing the size allocations during the updates, until the migration is complete. Their algorithm has polynomial computation time, but has slightly stronger consistency requirements than the model of Hong et al. [2013].

Lastly, Brandt et al. [2016a; 2016c] consider a modified migration problem by not fixing the new multi-commodity flow, but just its demands. If the final (and every intermediate) configuration has no congestion then the locations of the flows in the network do not matter. In scenarios with a single destination (or a single source), augmenting flows can be used to compute the individual updates: Essentially, the flows are changed along the routes of the augmenting flows, allowing for a linear number of updates for splittable flows with intermediate paths. The augmentation model cannot be extended to the general case of multi-source multi-destination network flows.

6.2.2. Complexity. The complexity of capacitated updates can roughly be summarized as follows: Problems involving splittable flows can be decided in polynomial time, while restrictions such as unsplittable flows or memory limits turn the problem NP-hard, see Table III. In a way, the capacitated update problems differs from related network update problems in that it is not always solvable in a consistent way. On the other hand, e.g., per-packet/flow consistency can always be maintained by a 2-phase commit, and loop-free updates for a single destination can always be performed in a linear number of updates.

One standard approach in recent work for flow migration is linear (splittable flows) or integer programing (unsplittable flows): With the number of intermediate configurations x as an input, it is checked if a consistent migration with x intermediate states exists. Should the answer be yes, then

Table III. Table summarizing decision problem results for flow migration.

Flow migration problem	Intermediate paths	Memory restrictions	Decision problem hardness
Unsplittable	Yes	Yes	NP-hard [Brandt et al. 2016b]
		No	
	No	Yes	NP-hard [Förster and Wattenhofer 2016]
		No	
Unit size	Yes	Yes	NP-hard [Brandt et al. 2016b]
		No	
	No	Yes	Open (also for integer size)
		No	
Splittable	Yes	Yes	NP-hard [Jin et al. 2014]
		No	P [Brandt et al. 2016b]
	No	Yes	NP-hard [Jin et al. 2014]
		No	Open
Move every flow only once	Yes	Yes	Not allowed (model)
		No	
	No	Yes	NP-complete [Jin et al. 2014]
		No	NP-complete [Förster and Wattenhofer 2016]

Note: In general, it is unknown if flow migration is in NP if flows can be moved more than once, except for the case of splittable flows without memory restrictions. We note that if a problem is NP-hard without memory restrictions, it is also NP-hard with memory restrictions.

one can use a binary search over x to find the fastest schedule. This idea originated in *SWAN* [Hong et al. 2013] for splittable flows, and was later extended to other models, cf. Table II.

However, the LP-approach via binary search (likewise for the integer one) suffers from the drawback that it is only complete if the model is restricted: If x is unbounded, then one can only decide whether a migration with x updates exists, but not whether there is no migration schedule with y steps, for some $y > x$. Additionally, it is not even clear to what complexity class the general capacitated update problem belongs to, cf. the decision problem hardness column of Table III.

The only exception arises in case of splittable flows without memory restrictions, where either an (implicit) schedule or a certificate that no consistent migration is possible, is found in polynomial time [Brandt et al. 2016b]. The authors use a combinatorial approach not relying on linear programming. Adding memory restrictions turns this problem NP-hard as well [Jin et al. 2014].

If the model is restricted to allow every flow only to be moved once (from the old path to the new path), then the capacitated update problem becomes NP-complete [Förster and Wattenhofer 2016; Jin et al. 2014]: Essentially, as the number of updates is limited by the number of flows, the problem is in NP. In this specific case, one can also approximate the minimum congestion for unsplittable flows in polynomial time by randomized rounding [Zheng et al. 2015].

Hardly any (in-)approximability results exist today, and most work relies on reductions from the Partition problem, cf. Table IV. The only result that we are aware of is via a reduction from MAX 3-SAT, which also applies to unit size flows [Brandt et al. 2016b].

6.3. Related Optimization Problems

In a practical setting, splitting flows is often realized via deploying multiple unsplittable paths, which is an NP-hard optimization problem as well, both for minimizing the number of paths and for maximizing k-splittable flows, cf. [Baier et al. 2005; Hartman et al. 2012]. Another popular option is to split the flows at the routers using hash functions; other major techniques are flow(let) caches and round-robin splitting, cf. [He and Rexford 2008]. Nonetheless, splitting flows along multiple paths can lead to packet reordering problems, which need to be handled by further techniques, see, e.g., [Kandula et al. 2007].

Many of the discussed flow migration works rely on linear programming formulations: Even though their runtime is polynomial in theory, the timely migration of large networks with many intermediate states is currently problematic in practice [Hong et al. 2013]. If the solution takes too long to compute, the to-be solved problem might no longer exist, a problem only made worse when when resorting to (NP-hard) integer programming for unsplittable flows. As such, some tradeoff has to be made between finding an optimal solution and one that can actually be deployed.

Table IV. Compact overview of flow migration hardness techniques and results.

Ref.	Reduction via	(Un-)splittable model	Interm. paths	Memory limits	Decision prob. in general	Optimization problems/remarks
[Jin et al. 2014]	Partition	Splittable	No	Yes	NP-hard	NP-complete if every flow may only move once
[Jin et al. 2014]	Partition	Splittable	No	No	–	NP-hard (fewest rule modifications)
[Brandt et al. 2016b]	–	Splittable	Yes	No	P	Fastest schedule can be of unbounded length, LP for new reachable demands if cannot migrate
[Förster and Wattenhofer 2016]	–	2-Splittable	No	No	P	studies slightly different model
[Brandt et al. 2016b]	(MAX) 3-SAT	Unsplittable	Yes	No	NP-hard (also for unit size)	NP-hard to approx. additive error of flow removal for consistency better than $7/8 + \varepsilon$
[Zheng et al. 2015]	Partition	Unsplittable	Yes/No	No	–	NP-hard (fastest schedule)
[Förster and Wattenhofer 2016]	Partition	Unsplittable	No	No	NP-hard	stronger consistency model, but proof carries over
[Luo et al. 2016]	Partition & Subset Sum	Unsplittable	No	No	–	NP-hard for 3-update schedule

Orthogonal to the problem of consistent flow migration is the approach of scheduling flows beforehand, not changing their path assignments in the network during the update. We refer to the recent works by Kandula et al. [2014] and Perry et al. [2014] for examples. Game-theoretic approaches have also been considered, e.g., [Hoefer et al. 2011].

Lastly, the application of model checking to consistent network updates does not cover bandwidth problem restrictions yet [McClurg et al. 2015; Zhou et al. 2015].

6.4. Concluding Remarks and Open Problems

The classification of the complexity of flow migration still poses many questions, cf. Table IV: If every flow can only be moved once, then the migration (decision) problem is clearly in NP. However, what is the decision complexity if flows can be moved arbitrarily often, especially with intermediate paths? Is the "longest" fastest update schedule for unsplittable flows: linear, polynomial or exponential, or even worse? Related questions are also open for flows of unit or integer size in general.

The problem of migrating splittable flows without memory limits and without intermediate paths is still not studied either: It seems as if the methods of [Brandt et al. 2016b] and [Förster and Wattenhofer 2016] also apply to this case, but a formal proof is missing. Another open issue which researchers recently started to consider concerns how to exploit traffic engineering flexibilities and "helper rules" to jointly optimize update scheduling and route selection [Xu et al. 2016].

7. RELAXING SAFETY GUARANTEES

So far we studied network updates assuming that consistency in the respective model must be maintained, e.g., no forwarding loops must appear at any time. In situations where the computation is no longer tractable or the consistency property cannot be maintained at all, some works proposed to break consistency in a controlled manner.

A first approach in this direction consists in trying to minimize the time spent in an inconsistent state, with underlying protocols being able to correct the induced problems (e.g., dropped packets are re-transmitted), as done in a production environment in Google's *B4* [Jain et al. 2013].

An alternative is to synchronize the clocks in the switches so that network updates can be performed simultaneously: With perfect clock synchronization, lossless communications and switch execution behavior, loop freedom could be maintained. As the standard Network Time Protocol (NTP) does not have sufficient synchronization behavior, the Precision Time Protocol (PTP) was adapted to SDN environments in [Mizrahi and Moses 2014b; 2014c], achieving microsecond accuracy in experiments.

This obviously comes with additional message overhead for time synchronization in the whole network.

Nonetheless, in some situations synchronized updates can be considered optimal: E.g., consider the case in Fig. 11 where two unsplittable flows need to be swapped [Mizrahi and Moses 2014a], with no alternative paths in the network available for the final links. Then, synchronizing the new flow paths can minimize the induced congestion [Mizrahi and Moses 2016c]. Synchronized updates cannot guarantee packet consistency on their own, as packets that are currently en-route may still encounter changed forwarding rules at the next switch. Time can also be used similarly to a 2-phase commit though, by analogously using timestamps in the packet header as tags during the update [Mizrahi et al. 2015], with [Mizrahi et al. 2015] also showing an efficient implementation using timestamp-based TCAM ranges. Additional memory, as in the 2-phase commit approach of Reitblatt et al. [2012], will be used for this method, but packets only need to be tagged implicitly by including the timestamp (where often 1 bit suffices [Mizrahi and Moses 2016a; Mizrahi et al. 2015]). In [Mizrahi and Moses 2013] some additional methods are discussed on how to guarantee packet consistency by temporarily storing traffic at the switches.

Despite all those advantages, the proposed clock synchronization approaches do not prevent unpredictable variations of command execution time on network switches [Jin et al. 2014], motivating the need for prediction-based scheduling methods [Mizrahi and Moses 2016b; 2016d]. Even worse, failures have an intrinsic, unavoidable cost in this approach. If a switch fails to update at all, the network can stay in an inconsistent state until the controller is notified and takes appropriate actions (e.g., computing another update). The same risk of inconsistencies holds if controller-to-switch messages are delayed or lost. In contrast, techniques based on sequential approaches can verify the application of sent update commands one by one, possibly moving forward (to the next update) or back (if a command is not received or not yet applied) with no risk of incurring safety violations.

8. FROM THEORY TO PRACTICE

As a complement to the previously-described theoretical and algorithmic results, we now provide an overview on practical challenges to ensure consistent network updates. We also describe how previous works tackled those challenges in order to build automated systems that can automatically carry out consistent updates.

(1) **Ensuring basic communication with network devices:** Automated update systems classically rely on a logically-centralized coordinator, which must interact with network devices to instruct them to apply operations (in a given order). Such a device-coordinator interaction requires a communication channel. Update coordinators in traditional networks typically exploit the command line interface of devices [Chen et al. 2009; Vanbever et al. 2011]. For SDNs, the interaction is simplified by their very architecture, since the coordinator is typically embodied by the SDN controller which must be already able to program (e.g., through OpenFlow [McKeown et al. 2008] or similar protocols) and monitor (e.g., thanks to a Network Information Base [Koponen et al. 2010]) the controlled devices.

(2) **Applying operational sequences, step by step:** Both devices and the device-coordinator communication are not necessarily reliable. For example, messages sent by the coordinator may be lost or not be applied by all devices upon reception [Jin et al. 2014; Kuzniar et al. 2014]. Those possibilities are typically taken into account in the computation of the update sequence (see §3). However, an effective update system must also ensure that operations are actually applied as in the computed sequences, e.g., that all operations in one update step are actually executed on the switches before sending operations in the next step. To this end, a variety of strategies are applied in the literature, from dedicated monitoring approaches (based on available network primitives like status-checking commands and protocols [Chen et al. 2009], lower-level packet cloning mechanisms [Vanbever et al. 2011], or active probing packets [Peresíni et al. 2015]) to acknowledgement-based protocols implemented by SDN devices [Kuzniar et al. 2014].

(3) **Working around device limitations:** Applying carefully-computed operational sequences ensures update consistency but not necessarily performance (e.g., speed), as the latter also depends on device efficiency in executing operations. This aspect has been analyzed by several works, especially focused on SDN updates which are more likely to be applied in real-time (e.g., even to react to a failure). It has been pointed out that current SDN device limitations impact update performance in two ways. First, SDN switches are not yet fast to change their packet-processing rules, as highlighted by several measurement studies. For example, in the Devoflow [Curtis et al. 2011] paper, the authors showed that the rate of statistics gathering is limited by the size of the flow table and is negatively impacted by the flow setup rate. In 2015, He et al. [2015] experimentally demonstrated the high rule installation latency of four different types of production SDN switches. This confirmed the results of independent studies [Huang et al. 2013; Rotsos et al. 2012] providing a more in-depth look into switch performance across various vendors. Second, rule installation time can highly vary over time, independently on any switch, because it is a function of runtime factors like already-installed rules and data-plane load. The measurement campaign on real OpenFlow switches performed in Dionysus [Jin et al. 2014] indeed shows that rule installation delay can vary from seconds to minutes. Update systems are therefore engineered to mitigate the impact of those limitations – despite not avoiding per-rule update bottlenecks. Prominently, Dionysus [Jin et al. 2014] significantly reduces multi-switch update latency by carefully scheduling operations according to dynamic switch conditions. In addition, CoVisor [Jin et al. 2015] and [Dudycz et al. 2016] minimize the number of rule updates sent to switches through eliminating redundant updates.

(4) **Avoiding conflicts between multiple control-planes:** For availability, performance, and robustness, network control-planes are often physically-distributed, even when logically centralized (as in the case of SDNs with replicated controllers). For updates of traditional networks, the control-plane distribution is straightforwardly taken into account, since it is encompassed in the update problem definition (see §2). In contrast, additional care must be applied to SDN networks with multiple controllers: if several controllers try to update network devices at the same time, one controller may override rules installed by another, impacting the correctness of the update (both during and after the update itself). This requires to solve potential conflicts between controllers, e.g., by pro-actively specifying how the final rules have to computed (e.g., [Monsanto et al. 2013]), by implementing coordination and locking primitives on switches (e.g., [Schiff et al. 2016]), or by reactively detecting and possibly resolving conflicts (e.g., [Canini et al. 2015]). A generalization of the above setting consists in considering multiple control-planes that may be either all distributed, all centralized, or mixed (some distributed and some centralized). Potential conflicts and general meta-algorithms to ensure consistent updates in those cases are described in [Vissicchio et al. 2015].

(5) **Updating the control-plane:** In traditional networks, data-plane changes can only be enforced by changing the configuration of control-plane protocols (e.g., IGPs). In contrast, the most studied case for SDN updates considers an unmodified controller that has to change the packet-processing rules on network switches. Nevertheless, a few works also considered the problem of entirely replacing the SDN controller itself, e.g., upgrading it to a new version or replacing an old controller with a newer one. In particular, HotSwap [Vanbever et al. 2013a] describes an architecture that enables the replacement of an SDN controller, by relying on a hypervisor that maintains a history of network events. As an alternative, explicit state transfer is used to design and implement the Morpheus controller platform in [Saur et al. 2016].

(6) **Dealing with events occurring during an update:** Operational sequences computed by update algorithms forcedly assume stable network conditions. In practice, however, unpredictable events, like failures, can modify the network behavior concurrently and independently from the operations performed to update the network. While concurrent events can be very unlikely (especially for fast updates), by definition they cannot be prevented. A few contributions assessed the impact of such unpredictable events on the update safety. For instance, the impact of link failures on SITN-based IGP reconfigurations is experimentally evaluated in [Vanbever et al.

2012]. A more systematic approach is taken by the recent FOUM work [Hua et al. 2016], that aims at guaranteeing per-packet consistency in the presence of an adversary able to perform packet-tampering and packet-dropping attacks.

9. FUTURE RESEARCH DIRECTIONS

While we have already identified specific open research questions in the corresponding sections, we now discuss more general areas which we believe deserve more attention by the research community in the future.

(1) **Charting the complexity landscape:** Researchers have only started to understand the computational complexities underlying the network update problem. In particular, many NP-hardness results have been derived for general problem formulations for all three of our consistency models: connectivity consistency, policy consistency, and performance consistency. So far, only for a small number of specific models polynomial-time optimal algorithms are known. Even less is known about approximation algorithms. Hence, more research efforts would be needed to chart a clearer picture of the complexity landscape for network update problems. We expect that some of these insights will also have interesting implications on classic optimization problems.
(2) **Refining our models:** While today's network models capture well the fundamental constraints and tradeoffs in consistent network update problems, these models are still relatively simple. In our opinion, more refined models can be developed, for example to take into account additional performance aspects (e.g., the impact of packet reorderings on throughput). Moreover, new models could better leverage empirical knowledge about networks. For example, they can include assumptions on the upper and lower bound of switch update times and delays of the channel between SDN controller and controlled switches.
(3) **Considering new update problems:** We expect future update techniques to ensure consistency of higher-level network requirements (like NFV, path delay, etc.), the same way as recent SDN controllers are supporting them. More dynamic and stateful applications also introduce new consistency criteria (e.g., [Ghorbani and Godfrey 2014; McClurg et al. 2016]) that might be interesting to take into account in future update techniques.
(4) **Dealing with distributed control planes:** We believe that researchers have only started to understand the design and implication of SDN control planes which are distributed (not only vertically but also horizontally [Phemius et al. 2014; Nguyen et al. 2016; Schmid and Suomela 2013]), e.g., for dependability and performance purposes. In particular, distributed SDN control-planes introduce additional challenges in terms of consistent network updates and controller coordination.

10. CONCLUSION

The purpose of this survey was to provide researchers active in or interested in the field of network update problems with an overview of the state-of-the-art, including models, techniques, impossibility results as well as practical challenges. We also presented a historical perspective and discussed the fundamental new challenges introduced in Software-Defined Networks, also relating them to classic graph-theoretic optimization problems. Finally, we have identified open questions for future research.

ACKNOWLEDGMENTS

For inputs and feedback, we thank Marco Canini, Nate Foster, Dejan Kostić, Ratul Mahajan, Roger Wattenhofer, and Jiaqi Zheng.

REFERENCES

Richard Alimi, Ye Wang, and Yang Richard Yang. 2008. Shadow configuration as a network management primitive. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, Victor Bahl, David Wetherall, Stefan Savage, and Ion Stoica (Eds.). ACM, 111–122. DOI:http://dx.doi.org/10.1145/1402958.1402972

Saeed Amiri, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. 2016. Transiently Consistent SDN Updates: Being Greedy is Hard. In *Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2016*.

Georg Baier, Ekkehard Köhler, and Martin Skutella. 2005. The k-Splittable Flow Problem. *Algorithmica* 42, 3-4 (2005), 231–248. DOI: http://dx.doi.org/10.1007/s00453-005-1167-9

Barefoot Networks. 2016. The world's fastest and most programmable networks (white paper). https://barefootnetworks.com/white-paper/the-worlds-fastest-most-programmable-networks/. (2016).

Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: programming protocol-independent packet processors. *Computer Communication Review* 44, 3 (2014), 87–95. DOI: http://dx.doi.org/10.1145/2656877.2656890

Sebastian Brandt, Klaus-Tycho Förster, and Roger Wattenhofer. 2016a. Augmenting anycast network flows. In *Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, January 4-7, 2016*. ACM, 24:1–24:10. DOI: http://dx.doi.org/10.1145/2833312.2833450

Sebastian Brandt, Klaus-Tycho Förster, and Roger Wattenhofer. 2016b. On consistent migration of flows in SDNs. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524332

Sebastian Brandt, Klaus-Tycho Förster, and Roger Wattenhofer. 2016c. Augmenting Flows for the Consistent Migration of Multi-Commodity Single-Destination Flows in SDNs. *Pervasive and Mobile Computing* (to appear, available online) (2016). DOI: http://dx.doi.org/10.1016/j.pmcj.2016.09.012

Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. 2015. A distributed and robust SDN control plane for transactional network updates. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*. 190–198. DOI: http://dx.doi.org/10.1109/INFOCOM.2015.7218382

Martín Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: taking control of the enterprise. In *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007*, Jun Murai and Kenjiro Cho (Eds.). ACM, 1–12. DOI: http://dx.doi.org/10.1145/1282380.1282382

Pavol Cerný, Nate Foster, Nilesh Jagnik, and Jedidiah McClurg. 2016. Optimal Consistent Network Updates in Polynomial Time. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings (Lecture Notes in Computer Science)*, Cyril Gavoille and David Ilcinkas (Eds.), Vol. 9888. Springer, 114–128. DOI: http://dx.doi.org/10.1007/978-3-662-53426-7_9

Xu Chen, Zhuoqing Morley Mao, and Jacobus E. van der Merwe. 2009. PACMAN: a platform for automated and controlled network operations and configuration management. In *Proceedings of the 2009 ACM Conference on Emerging Networking Experiments and Technology, CoNEXT 2009, Rome, Italy, December 1-4, 2009*, Jörg Liebeherr, Giorgio Ventre, Ernst W. Biersack, and Srinivasan Keshav (Eds.). ACM, 277–288. DOI: http://dx.doi.org/10.1145/1658939.1658971

François Clad, Pascal Mérindol, Jean-Jacques Pansiot, Pierre François, and Olivier Bonaventure. 2014. Graceful Convergence in Link-State IP Networks: A Lightweight Algorithm Ensuring Minimal Operational Impact. *IEEE/ACM Trans. Netw.* 22, 1 (2014), 300–312. DOI: http://dx.doi.org/10.1109/TNET.2013.2255891

François Clad, Pascal Mérindol, Stefano Vissicchio, Jean-Jacques Pansiot, and Pierre François. 2013. Graceful router updates in link-state protocols. In *2013 21st IEEE International Conference on Network Protocols, ICNP 2013, Göttingen, Germany, October 7-10, 2013*. IEEE, 1–10. DOI: http://dx.doi.org/10.1109/ICNP.2013.6733587

François Clad, Stefano Vissicchio, Pascal Mérindol, Pierre François, and Jean-Jacques Pansiot. 2015. Computing Minimal Update Sequences for Graceful Router-Wide Reconfigurations. *IEEE/ACM Trans. Netw.* 23, 5 (2015), 1373–1386. DOI: http://dx.doi.org/10.1109/TNET.2014.2332101

Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. DevoFlow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul (Eds.). ACM, 254–265. DOI: http://dx.doi.org/10.1145/2018436.2018466

Szymon Dudycz, Arne Ludwig, and Stefan Schmid. 2016. Can't Touch This: Consistent Network Updates for Multiple Policies. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016*. IEEE Computer Society, 133–143. DOI: http://dx.doi.org/10.1109/DSN.2016.21

Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. 1998. Approximating Minimum Feedback Sets and Multicuts in Directed Graphs. *Algorithmica* 20, 2 (1998), 151–174. DOI: http://dx.doi.org/10.1007/PL00009191

Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. 2013. FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, Nate Foster and Rob Sherwood (Eds.). ACM, 19–24. DOI: http://dx.doi.org/10.1145/2491185.2491203

Nick Feamster, Jennifer Rexford, and Ellen W. Zegura. 2014. The road to SDN: an intellectual history of programmable networks. *Computer Communication Review* 44, 2 (2014), 87–98. DOI: http://dx.doi.org/10.1145/2602204.2602219

Klaus-Tycho Förster, Ratul Mahajan, and Roger Wattenhofer. 2016. Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/IFIPNetworking.2016.7497232

Klaus-Tycho Förster and Roger Wattenhofer. 2016. The Power of Two in Consistent Network Updates: Hard Loop Freedom, Easy Flow Migration. In *25th International Conference on Computer Communication and Networks, ICCCN 2016, Waikoloa, HI, USA, August 1-4, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/ICCCN.2016.7568583

Nate Foster, Arjun Guha, Mark Reitblatt, Alec Story, Michael J. Freedman, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Jennifer Rexford, Cole Schlesinger, David Walker, and Rob Harrison. 2013. Languages for software-defined networks. *IEEE Communications Magazine* 51, 2 (2013), 128–134. DOI: http://dx.doi.org/10.1109/MCOM.2013.6461197

Pierre François and Olivier Bonaventure. 2007. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Trans. Netw.* 15, 6 (2007), 1280–1292. DOI: http://dx.doi.org/10.1145/1373476.1373482

Pierre François, Olivier Bonaventure, Bruno Decraene, and Pierre-Alain Coste. 2007a. Avoiding Disruptions During Maintenance Operations on BGP Sessions. *IEEE Trans. Network and Service Management* 4, 3 (2007), 1–11. DOI: http://dx.doi.org/10.1109/TNSM.2007.021102

Pierre François, Mike Shand, and Olivier Bonaventure. 2007b. Disruption Free Topology Reconfiguration in OSPF Networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA*. IEEE, 89–97. DOI: http://dx.doi.org/10.1109/INFCOM.2007.19

Jing Fu, Peter Sjödin, and Gunnar Karlsson. 2008. Loop-free updates of forwarding tables. *IEEE Trans. Network and Service Management* 5, 1 (2008), 22–35. DOI: http://dx.doi.org/10.1109/TNSM.2008.080103

Soudeh Ghorbani and Brighten Godfrey. 2014. Towards correct network virtualization. In *2014 Workshop on Hot topics in software defined networking, HotSDN '14, Chicago, Illinois, USA, August 22, 2014*. 109–114. DOI: http://dx.doi.org/10.1145/2620728.2620754

Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. 2012. How to split a flow?. In *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*, Albert G. Greenberg and Kazem Sohraby (Eds.). IEEE, 828–836. DOI: http://dx.doi.org/10.1109/INFCOM.2012.6195830

Jiayue He and Jennifer Rexford. 2008. Toward internet-wide multipath routing. *IEEE Network* 22, 2 (2008), 16–21. DOI: http://dx.doi.org/10.1109/MNET.2008.4476066

Keqiang He, Junaid Khalid, Aaron Gember-Jacobson, Sourav Das, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. 2015. Measuring control plane latency in SDN-enabled switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15, Santa Clara, California, USA, June 17-18, 2015*, Jennifer Rexford and Amin Vahdat (Eds.). ACM, 25:1–25:6. DOI: http://dx.doi.org/10.1145/2774993.2775069

Gonzalo Herrero and Jan van der Ven. 2010. *Network Mergers and Migrations: Junos Design and Implementation.* Wiley.

Martin Hoefer, Vahab S. Mirrokni, Heiko Röglin, and Shang-Hua Teng. 2011. Competitive routing over time. *Theor. Comput. Sci.* 412, 39 (2011), 5420–5432. DOI: http://dx.doi.org/10.1016/j.tcs.2011.05.055

Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving High Utilization with Software-driven WAN. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 15–26. DOI: http://dx.doi.org/10.1145/2534169.2486012

Jingyu Hua, Xin Ge, and Sheng Zhong. 2016. FOUM: A flow-ordered consistent update mechanism for software-defined networking in adversarial settings. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524499

Danny Yuxing Huang, Ken Yocum, and Alex C. Snoeren. 2013. High-fidelity switch models for software-defined network emulation. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, Nate Foster and Rob Sherwood (Eds.). ACM, 43–48. DOI: http://dx.doi.org/10.1145/2491185.2491188

Mark Imbriaco. 2012. Network problems last Friday. GitHub: https://github.com/blog/1346networkproblemslastfriday. (December 2012).

J. Jackson. 2012. Godaddy blames outage on corrupted router tables. PCWorld: http://www.pcworld.com/article/262142/godaddy_blames_outage_on_corrupted_router_tables.html. (September 2012).

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 3–14. DOI: http://dx.doi.org/10.1145/2534169.2486019

Xin Jin, Jennifer Gossels, Jennifer Rexford, and David Walker. 2015. CoVisor: A Compositional Hypervisor for Software-Defined Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*. USENIX Association, 87–101. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/jin

Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. 2016. Optimizing Bulk Transfers with Software-Defined Optical WAN. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016*

Conference, Florianopolis, Brazil, August 22-26, 2016, Marinho P. Barcellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti (Eds.). ACM, 87–100. DOI: http://dx.doi.org/10.1145/2934872.2934904

Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. 2014. Dynamic Scheduling of Network Updates. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 539–550. DOI: http://dx.doi.org/10.1145/2740070.2626307

John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas E. Anderson, and Arun Venkataramani. 2008. Consensus Routing: The Internet as a Distributed System. (Best Paper). In *5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008, April 16-18, 2008, San Francisco, CA, USA, Proceedings*, Jon Crowcroft and Michael Dahlin (Eds.). USENIX Association, 351–364. http://www.usenix.org/events/nsdi08/tech/full_papers/john/john.pdf

Marcin Kaminski, Paul Medvedev, and Martin Milanic. 2011. Shortest paths between shortest paths. *Theor. Comput. Sci.* 412, 39 (2011), 5205–5210.

Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. 2007. Dynamic load balancing without packet reordering. *SIGCOMM Comput. Commun. Rev.* 37, 2 (2007), 51–62. DOI: http://dx.doi.org/10.1145/1232919.1232925

Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. 2014. Calendaring for Wide Area Networks. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 515–526. DOI: http://dx.doi.org/10.1145/2740070.2626336

Naga Praveen Katta, Jennifer Rexford, and David Walker. 2013. Incremental consistent updates. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, Nate Foster and Rob Sherwood (Eds.). ACM, 49–54. DOI: http://dx.doi.org/10.1145/2491185.2491191

Peyman Kazemian, Michael Chan, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real Time Network Policy Checking Using Header Space Analysis. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, Nick Feamster and Jeffrey C. Mogul (Eds.). USENIX Association, 99–111. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/kazemian

Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, Steven D. Gribble and Dina Katabi (Eds.). USENIX Association, 113–126. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian

Eric Keller, Jennifer Rexford, and Jacobus E. van der Merwe. 2010. Seamless BGP Migration with Router Grafting. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*. USENIX Association, 235–248. http://www.usenix.org/events/nsdi10/tech/full_papers/keller.pdf

Ram Keralapura, Chen-Nee Chuah, and Yueyue Fan. 2006. Optimal Strategy for Graceful Network Upgrade. In *Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management (INM '06)*. ACM, New York, NY, USA, 83–88. DOI: http://dx.doi.org/10.1145/1162638.1162652

Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and Brighten Godfrey. 2012. Veriflow: verifying network-wide invariants in real time. *Computer Communication Review* 42, 4 (2012), 467–472. DOI: http://dx.doi.org/10.1145/2377677.2377766

Teemu Koponen, Martín Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. 2010. Onix: A Distributed Control Platform for Large-scale Production Networks. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, Remzi H. Arpaci-Dusseau and Brad Chen (Eds.). USENIX Association, 351–364. http://www.usenix.org/events/osdi10/tech/full_papers/Koponen.pdf

Nate Kushman, Srikanth Kandula, Dina Katabi, and Bruce M. Maggs. 2007. R-BGP: Staying Connected in a Connected World. In *4th Symposium on Networked Systems Design and Implementation (NSDI 2007), April 11-13, 2007, Cambridge, Massachusetts, USA, Proceedings.*, Hari Balakrishnan and Peter Druschel (Eds.). USENIX. http://www.usenix.org/events/nsdi07/tech/kushman.html

Maciej Kuzniar, Peter Peresíni, and Dejan Kostic. 2014. Providing Reliable FIB Update Acknowledgments in SDN. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, Aruna Seneviratne, Christophe Diot, Jim Kurose, Augustin Chaintreau, and Luigi Rizzo (Eds.). ACM, 415–422. DOI: http://dx.doi.org/10.1145/2674005.2675006

Franck Le, Geoffrey G. Xie, Dan Pei, Jia Wang, and Hui Zhang. 2008. Shedding light on the glue logic of the internet routing architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, Victor Bahl, David Wetherall, Stefan Savage, and Ion Stoica (Eds.). ACM, 39–50. DOI: http://dx.doi.org/10.1145/1402958.1402964

Michael Lewin, Dror Livnat, and Uri Zwick. 2002. Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems. In *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings (Lecture Notes in Computer Science)*, William J. Cook and Andreas S. Schulz (Eds.), Vol. 2337. Springer, 67–82. http://link.springer.de/link/service/series/0558/bibs/2337/23370067.htm

Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer, and David Maltz. 2013. zUpdate: Updating Data Center Networks with Zero Loss. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 411–422. DOI: http://dx.doi.org/10.1145/2534169.2486005

Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid. 2016. Transiently Secure Network Updates. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science, Antibes Juan-Les-Pins, France, June 14-18, 2016*, Sara Alouf, Alain Jean-Marie, Nidhi Hegde, and Alexandre Proutière (Eds.). ACM, 273–284. DOI: http://dx.doi.org/10.1145/2896377.2901476

Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. 2015. Scheduling Loop-free Network Updates: It's Good to Relax!. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, Chryssis Georgiou and Paul G. Spirakis (Eds.). ACM, 13–22. DOI: http://dx.doi.org/10.1145/2767386.2767412

Arne Ludwig, Matthias Rost, Damien Foucard, and Stefan Schmid. 2014. Good Network Updates for Bad Packets: Waypoint Enforcement Beyond Destination-Based Routing Policies. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks, HotNets-XIII, Los Angeles, CA, USA, October 27-28, 2014*, Ethan Katz-Bassett, John S. Heidemann, Brighten Godfrey, and Anja Feldmann (Eds.). ACM, 15:1–15:7. DOI: http://dx.doi.org/10.1145/2670518.2673873

Long Luo, Hongfang Yu, Shouxi Luo, and Mingui Zhang. 2015. Fast lossless traffic migration for SDN updates. In *2015 IEEE International Conference on Communications, ICC 2015, London, United Kingdom, June 8-12, 2015*. IEEE, 5803–5808. DOI: http://dx.doi.org/10.1109/ICC.2015.7249247

Shouxi Luo, Hong-Fang Yu, Long Luo, and Lemin Li. 2016. Arrange your network updates as you wish. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*. IEEE, 10–18. DOI: http://dx.doi.org/10.1109/IFIPNetworking.2016.7497214

Ratul Mahajan and Roger Wattenhofer. 2013. On consistent updates in software defined networks. In *Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, College Park, MD, USA, November 21-22, 2013*, Dave Levine, Sachin Katti, and Dave Oran (Eds.). ACM, 20:1–20:7. DOI: http://dx.doi.org/10.1145/2535771.2535791

Jedidiah McClurg, Hossein Hojjat, Pavol Cerný, and Nate Foster. 2015. Efficient synthesis of network updates. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, David Grove and Steve Blackburn (Eds.). ACM, 196–207. DOI: http://dx.doi.org/10.1145/2737924.2737980

Jedidiah McClurg, Hossein Hojjat, Nate Foster, and Pavol Cerný. 2016. Event-driven network programming. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*. 369–385. DOI: http://dx.doi.org/10.1145/2908080.2908097

Rick McGeer. 2012. A Safe, Efficient Update Protocol for Openflow Networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*. ACM, New York, NY, USA, 61–66. DOI: http://dx.doi.org/10.1145/2342441.2342454

Rick McGeer. 2013. A correct, zero-overhead protocol for network updates. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, Nate Foster and Rob Sherwood (Eds.). ACM, 161–162. DOI: http://dx.doi.org/10.1145/2491185.2491217

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. DOI: http://dx.doi.org/10.1145/1355734.1355746

Martin Middendorf. 1994. Supersequences, runs, and CD grammar systems. *Developments in Theoretical Computer Science* 6 (1994), 101–114.

Tal Mizrahi and Yoram Moses. 2013. Time-based updates in software defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, Nate Foster and Rob Sherwood (Eds.). ACM, 163–164. DOI: http://dx.doi.org/10.1145/2491185.2491214

Tal Mizrahi and Yoram Moses. 2014a. On the Necessity of Time-based Updates in SDN. In *Open Networking Summit 2014 - Research Track, ONS 2014, Santa Clara, CA, USA, March 2-4, 2014*, Rob Sherwood (Ed.). USENIX Association. https://www.usenix.org/conference/ons2014/technical-sessions/presentation/mizrahi

Tal Mizrahi and Yoram Moses. 2014b. ReversePTP: a software defined networking approach to clock synchronization. In *Proceedings of the third workshop on Hot topics in software defined networking, HotSDN '14, Chicago, Illinois, USA, August 22, 2014*, Aditya Akella and Albert G. Greenberg (Eds.). ACM, 203–204. DOI: http://dx.doi.org/10.1145/2620728.2620764

T. Mizrahi and Y. Moses. 2014c. Using ReversePTP to distribute time in Software Defined Networks. In *2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. 112–117. DOI: http://dx.doi.org/10.1109/ISPCS.2014.6948702

Tal Mizrahi and Yoram Moses. 2016a. The case for Data Plane Timestamping in SDN. In *IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 856–861. DOI: http://dx.doi.org/10.1109/INFCOMW.2016.7562197

Tal Mizrahi and Yoram Moses. 2016b. OneClock to rule them all: Using time in networked applications. In *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, Istanbul, Turkey, April 25-29, 2016*, Sema Oktug, Mehmet Ulema, Cicek Cavdar, Lisandro Zambenedetti Granville, and Carlos Raniery Paula dos Santos (Eds.). IEEE, 679–685. DOI: http://dx.doi.org/10.1109/NOMS.2016.7502876

Tal Mizrahi and Yoram Moses. 2016c. Software defined networks: It's about time. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524418

Tal Mizrahi and Yoram Moses. 2016d. Time Capability in NETCONF. In *RFC 7758*.

Tal Mizrahi, Ori Rottenstreich, and Yoram Moses. 2015. TimeFlip: Scheduling network updates with timestamp-based TCAM ranges. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*. IEEE, 2551–2559. DOI: http://dx.doi.org/10.1109/INFOCOM.2015.7218645

Ram Mohan. 2011. Storms in the cloud: Lessons from the amazon cloud outage. Security Week: http://www.securityweek.com/storms-cloud-lessons-amazon-cloud-outage. (June 2011).

Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. 2013. Composing Software Defined Networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, Nick Feamster and Jeffrey C. Mogul (Eds.). USENIX Association, 1–13. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/monsanto

J. Moy, P. Pillay-Esnault, and A. Lindem. 2003. Graceful OSPF Restart. RFC 3623. (2003).

Thanh Dang Nguyen, Marco Chiesa, and Marco Canini. 2016. Towards Decentralized Fast Consistent Updates. In *2016 ACM Applied Networking Research Workshop, ANRW*.

Stefano Paris, Apostolos Destounis, Lorenzo Maggi, Georgios S. Paschos, and Jérémie Leguay. 2016. Controlling flow reconfigurations in SDN. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524330

Ivan Pepelnjak. 2007. Changing the Routing Protocol in Your Network. ipSpace: http://blog.ipspace.net/2007/10/change-routing-protocol-in-your-network.html. (October 2007).

Peter Peresíni, Maciej Kuzniar, and Dejan Kostic. 2015. Rule-level Data Plane Monitoring With Monocle. In *2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*. 595–596. DOI: http://dx.doi.org/10.1145/2785956.2790012

Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A Centralized "Zero-queue" Datacenter Network. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 307–318. DOI: http://dx.doi.org/10.1145/2740070.2626309

Kevin Phemius, Mathieu Bouet, and Jeremie Leguay. 2014. DISCO: Distributed SDN controllers in a multi-domain environment. In *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*. 1–2. DOI: http://dx.doi.org/10.1109/NOMS.2014.6838273

Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying middlebox policy enforcement using SDN. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan (Eds.). ACM, 27–38. DOI: http://dx.doi.org/10.1145/2486001.2486022

Saqib Raza, Yuanchen Zhu, and Chen-Nee Chuah. 2009. Graceful Network Operations. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*. IEEE, 289–297. DOI: http://dx.doi.org/10.1109/INFCOM.2009.5061932

Saqib Raza, Yuanbo Zhu, and Chen-Nee Chuah. 2011. Graceful network state migrations. *IEEE/ACM Trans. Netw.* 19, 4 (2011), 1097–1110. DOI: http://dx.doi.org/10.1109/TNET.2010.2097604

Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. 2012. Abstractions for network update. In *ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012*, Lars Eggert, Jörg Ott, Venkata N. Padmanabhan, and George Varghese (Eds.). ACM, 323–334. DOI: http://dx.doi.org/10.1145/2342356.2342427

Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. 2011. Consistent updates for software-defined networks: change you can believe in!. In *Tenth ACM Workshop on Hot Topics in Networks (HotNets-X), HOTNETS '11, Cambridge, MA, USA - November 14 - 15, 2011*, Hari Balakrishnan, Dina Katabi, Aditya Akella, and Ion Stoica (Eds.). ACM, 7. DOI: http://dx.doi.org/10.1145/2070562.2070569

Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W. Moore. 2012. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *Passive and Active Measurement - 13th International Conference, PAM 2012, Vienna, Austria, March 12-14th, 2012. Proceedings (Lecture Notes in Computer Science)*, Nina Taft and Fabio Ricciato (Eds.), Vol. 7192. Springer, 85–95. DOI: http://dx.doi.org/10.1007/978-3-642-28537-0_9

Karla Saur, Joseph M. Collard, Nate Foster, Arjun Guha, Laurent Vanbever, and Michael W. Hicks. 2016. Safe and Flexible Controller Upgrades for SDNs. In *Proceedings of the Symposium on SDN Research, SOSR 2016, Santa Clara, CA, USA, March 14 - 15, 2016*, Brighten Godfrey and Martín Casado (Eds.). ACM, 8. DOI: http://dx.doi.org/10.1145/2890955.2890966

Liron Schiff, Stefan Schmid, and Petr Kuznetsov. 2016. In-Band Synchronization for Distributed SDN Control Planes. *Computer Communication Review* 46, 1 (2016), 37–43. DOI: http://dx.doi.org/10.1145/2875951.2875957

Stefan Schmid and Jukka Suomela. 2013. Exploiting locality in distributed SDN control. In *2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*. 121–126. DOI: http://dx.doi.org/10.1145/2491185.2491198

Aman Shaikh, Rohit Dube, and Anujan Varma. 2006. Avoiding instability during graceful shutdown of multiple OSPF routers. *IEEE/ACM Trans. Netw.* 14, 3 (2006), 532–542. DOI: http://dx.doi.org/10.1145/1143396.1143403

M. Shand and S. Bryant. 2010. *A Framework for Loop-Free Convergence*. RFC 5715. IETF.

M. Shand and L. Ginsberg. 2008. Restart Signaling for IS-IS. RFC 5306. (2008).

Lei Shi, Jing Fu, and Xiaoming Fu. 2009. Loop-Free Forwarding Table Updates with Minimal Link Overflow. In *Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14-18 June 2009*. IEEE, 1–6. DOI: http://dx.doi.org/10.1109/ICC.2009.5199146

United. 2011. United Airlines Restoring Normal Flight Operations Following Friday Computer Outage. http://newsroom.united.com/news-releases?item=124170. (June 2011).

Laurent Vanbever, Joshua Reich, Theophilus Benson, Nate Foster, and Jennifer Rexford. 2013a. HotSwap: correct and efficient controller upgrades for software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, Nate Foster and Rob Sherwood (Eds.). ACM, 133–138. DOI: http://dx.doi.org/10.1145/2491185.2491194

Laurent Vanbever, Stefano Vissicchio, Luca Cittadini, and Olivier Bonaventure. 2013b. When the cure is worse than the disease: The impact of graceful IGP operations on BGP. In *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. IEEE, 2220–2228. DOI: http://dx.doi.org/10.1109/INFCOM.2013.6567025

Laurent Vanbever, Stefano Vissicchio, Cristel Pelsser, Pierre François, and Olivier Bonaventure. 2011. Seamless network-wide IGP migrations. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul (Eds.). ACM, 314–325. DOI: http://dx.doi.org/10.1145/2018436.2018473

Laurent Vanbever, Stefano Vissicchio, Cristel Pelsser, Pierre François, and Olivier Bonaventure. 2012. Lossless migrations of link-state IGPs. *IEEE/ACM Trans. Netw.* 20, 6 (2012), 1842–1855. DOI: http://dx.doi.org/10.1109/TNET.2012.2190767

Stefano Vissicchio and Luca Cittadini. 2016. FLIP the (Flow) table: Fast lightweight policy-preserving SDN updates. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524419

Stefano Vissicchio, Luca Cittadini, Olivier Bonaventure, Geoffrey G. Xie, and Laurent Vanbever. 2015. On the co-existence of distributed and centralized routing control-planes. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*. IEEE, 469–477. DOI: http://dx.doi.org/10.1109/INFOCOM.2015.7218413

Stefano Vissicchio, Laurent Vanbever, Luca Cittadini, Geoffrey Xie, and Olivier Bonaventure. 2013. *Safe Update of Hybrid SDN Networks*. Technical Report. UCLouvain.

Stefano Vissicchio, Laurent Vanbever, Luca Cittadini, Geoffrey G. Xie, and Olivier Bonaventure. 2014. Safe routing reconfigurations with route redistribution. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*. IEEE, 199–207. DOI: http://dx.doi.org/10.1109/INFOCOM.2014.6847940

Stefano Vissicchio, Laurent Vanbever, Cristel Pelsser, Luca Cittadini, Pierre François, and Olivier Bonaventure. 2013. Improving Network Agility With Seamless BGP Reconfigurations. *IEEE/ACM Trans. Netw.* 21, 3 (2013), 990–1002. DOI: http://dx.doi.org/10.1109/TNET.2012.2217506

Wen Wang, Wenbo He, Jinshu Su, and Yixin Chen. 2016. Cupid: Congestion-free consistent data plane update in software defined networks. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524420

Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. 2008. Virtual Routers on the Move: Live Router Migration As a Network-management Primitive. *SIGCOMM Comput. Commun. Rev.* 38, 4 (Aug. 2008), 231–242. DOI: http://dx.doi.org/10.1145/1402946.1402985

Hongli Xu, Zhuolong Yu, Xiangyang Li, Chen Qian, and Liusheng Huang. 2016. Real-time Update with Joint Optimization on Route Selection and Update Scheduling for SDNs. In *2016 24th IEEE International Conference on Network Protocols, ICNP*.

Jiaqi Zheng, Hong Xu, Guihai Chen, and Haipeng Dai. 2015. Minimizing Transient Congestion during Network Update in Data Centers. In *23rd IEEE International Conference on Network Protocols, ICNP 2015, San Francisco, CA, USA, November 10-13, 2015*. IEEE Computer Society, 1–10. DOI: http://dx.doi.org/10.1109/ICNP.2015.33

Wenxuan Zhou, Dong (Kevin) Jin, Jason Croft, Matthew Caesar, and Philip Brighten Godfrey. 2015. Enforcing Customizable Consistency Properties in Software-Defined Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*. USENIX Association, 73–85. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/zhou