Safe Updates of Hybrid SDN Networks

Stefano Vissicchio*, Laurent Vanbever†, Luca Cittadini‡,
Geoffrey Xie§, Olivier Bonaventure*

* Université catholique de Louvain † Princeton University ‡ RomaTre University § Naval Postgraduate School

ABSTRACT

Software Defined Networking (SDN) promises to bring unparalleled flexibility, fine-grained control, configuration simplification and no vendor lock-in. The introduction of SDN in an existing network, however, must be incremental in most cases, for both technical and economical reasons. During the transition, operators have to manage hybrid networks, where SDN and traditional protocols coexist.

In this paper, we show that the simultaneous presence of SDN and traditional routing protocols can create forwarding anomalies that ultimately defeat the purpose of deploying SDN. We devise techniques to adapt traffic flows to network dynamics, update routing policies and incrementally deploy SDN in hybrid networks, while avoiding those anomalies. We assess the applicability of our approach by extensive simulations. By adding support for manageability and evolvability, our techniques make hybrid networks not only a means for transition but also an interesting design point that can merge advantages of SDN and traditional paradigms.

1. INTRODUCTION

By decoupling the data-plane from the control-plane, Software-Defined Networking (SDN) has enabled a lot of novel capabilities such as network virtualization [1], support for virtual-machine migration [2], flexible access control [3] and traffic engineering [4]. These new capabilities, along with the growing support for SDN protocols like OpenFlow [5], have increased the interest of the majority of network operators [6] to deploy SDN in a near future.

Despite all these advantages, network operators are usually (and understandably) slow to adopt new and disruptive network technologies. Replacing all the devices at once is just not possible considering the costs, the man power needed to perform the replacement, and the possible downtime. Instead, the transition is likely to be gradual, spanning several months (if not years).

During the transition, traditional devices will have to work along SDN devices in order to consistently forward packets. The cooperation between traditional and SDN devices can be established on the basis of the protocols supported. Indeed, device-specific controllers allow support of traditional protocols in SDN devices, as in RouteFlow [7]. Moreover, basic SDN capabilities (e.g., packet forwarding) can also be added to non SDN-capable devices by letting SDN controllers manipulate their static routes. In the following, we refer to networks in which traditional IGP and SDN protocols coexist by controlling different flows, as hybrid SDN networks, or simply *hybrid networks*. A peculiar example of running hybrid networks has been already reported in [8].

Besides facilitating the transition, hybrid networks can also be considered as a long-term objective in some networks. Indeed, hybrid networks enable operators to reap some of the benefits SDN without a full deployment. For example, very large networks like Internet Service Providers (ISP) usually have highly specific constraints that are unlikely to be fulfilled by SDN devices. Such constraints include the support of a wide variety of physical connectivity and the handling millions of forwarding entries [9]. As a comparison, SDN-enabled switches are mostly Ethernet-based and can only handle tens of thousands entries [10]. Partial deployments can also offload the SDN controller from tasks for which traditional routing protocols have been proven effective. For instance, network operators may need SDN fine-grained control capabilities only for a small percentage of flows, e.g., the few responsible for most of the traffic [11], or only at specific points in the network [8]. Traditional routing protocols also offer a simple solution for in-band connectivity between network devices and SDN controllers, or between SDN controllers handling different portions of the network.

While being unavoidable, hybrid networks are also harder to manage than pure IP or SDN networks. In particular, updating a hybrid network can trigger numerous forwarding inconsistencies due to the presence of multiple, potentially conflicting control-planes. Intuitively, a forwarding consistency occurs when parts of the traffic gets forwarded along unpredicted forwarding paths. In addition to disrupting routing policies (e.g., by bypassing a firewall) and traffic engineering policies (e.g., by forwarding along a backup link), these inconsistencies can also lead to forwarding loops and traffic

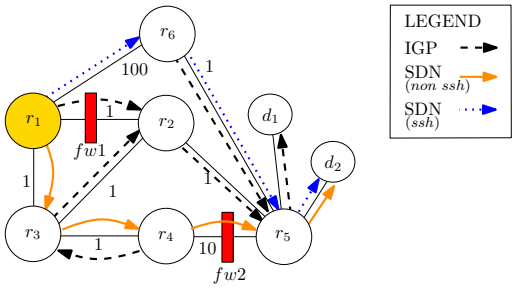


Figure 1: Example of hybrid network, in which a network update can trigger anomalies.

blackholes.

As an illustration, consider the network in Fig. 1, where network nodes are represented by circles, SDN-capable nodes by filled circles, and physical links by edges labeled with their IGP weights. Even if r_1 is the only SDN-enabled device, finer-grained policies than traditional networks are applied in the network. paths of IGP-controlled flows, while dotted blue arrows and solid orange arrows represent SDN-controlled flows for ssh traffic and the remaining traffic, respectively. Indeed, the IGP-controlled flows (dashed black arrows) from r_1 to d_1 use different paths than the SDN-controlled flows (solid orange arrows) from r_1 to d_2, even though both destinations are announced by r_5. The network contains two firewalls $fw1$ and $fw2$ that check all the traffic from r_1, except ssh traffic which is routed around them (dotted blue arrows). At a certain time, the operator may be willing to move the control of the traffic destined to d_1 from IGP to SDN, letting it follow the same path as the packets from r_1 to d_2. This can happen for multiple reasons, e.g., traffic engineering and better load balancing in response to traffic matrix changes, maintenance support for some links or devices in the path (r_1 r_2 r_5), or need to offload $fw1$. Such an update operation must be performed with care. For example, if r_1 is the first device to be updated, then a packet inconsistency occurs for packets from r_1 to d_1. In fact, r_1 will forward those packets to r_3, which has not been updated yet and will therefore forward traffic along the IGP path. The packets will thus bypass both firewalls $fw1$ and $fw2$, violating security policies. Also, a forwarding loop can occur if r_3 is reconfigured before r_4. The described anomalies will persist until other routers are reconfigured, hence potentially for a long time.

In this paper, we develop provably correct techniques that enable network operators to update hybrid networks *without* losing traffic or violating security policies. Our techniques enable: (i) any update of SDN-controlled forwarding paths; (ii) any update of the IGP-controlled forwarding paths, e.g., to arbitrarily change the weights of the IGP graph; (iii) any update in which SDN-controlled flows become IGP-controlled, and vice versa. Hence, our techniques can be leveraged to arbitrarily change forwarding paths of traffic flows, e.g., to adapt to the volatility of the heavy hitters [11] and to traffic trends (e.g., daylight) and to evolving requirements. We support all those scenarios by computing operational orderings which maintain network-wide consistency throughout the network update. Our techniques provide similar abstraction as the ones proposed recently by Reitblatt *et. al* [12]—while significantly improving their applicability and their scalability. Indeed, our techniques apply to any hybrid network (ranging from full IP to full SDN networks) and minimize the amount of duplicates entries required to perform the consistent update. As such, our techniques provide incentives both to start the transition and to consider hybrid networks as a target.

This paper makes the following contributions.

General model of hybrid networks. We introduce a formal model that captures the coexistence and interaction of different protocols in hybrid networks (§2). We make no assumptions on the type of IGP used in the hybrid networks and consider both Link-State (LS) and Distance-Vector (DV) IGPs. Since we assume the absence of BGP, our model directly applies to enterprise networks and to the many ISPs using MPLS. In the presence of BGP, our findings remain correct provided that the BGP configuration complies with known sufficient conditions for its correctness [13].

Theoretical guarantees. We prove that any-to-any connectivity can always be guaranteed during hybrid network updates (§3), either because of some specific IGP properties (for DV IGPs) or because of the reconfiguration technique itself (for LS IGPs).

Algorithms. Unlike connectivity, we show that it is not always possible to avoid path inconsistencies. We introduce the GPIA algorithm that finds the maximal sequence of safe operations, enabling the largest possible part of the network to be updated. We propose fallback solutions when no solution exists (§4).

Evaluation. We validate the effectiveness of GPIA on realistic traffic engineering reconfiguration scenarios, through extensive simulations. In most experiments, GPIA computed a safe operational ordering for the majority of the nodes (often more than 80%), hence allowing them to be reconfigured with no additional resource consumptions (e.g., no duplication of FIB entries).

Critical discussion. We discuss the relevance and generality of our theoretical results, as well as their limitations (§6). We compare with related work (§7), and describe future work and conclusions (§8)

2. MODEL AND PROBLEM STATEMENT

In this section, we report some background on control-

plane routing mechanisms (§2.1). Also, we describe our model for hybrid networks (§2.2), and we formally state the network update problem (§2.3).

2.1 Control-Plane Mechanisms

In a network, end-hosts data packets are relayed hop-by-hop by intermediate systems which we call network nodes, or simply *nodes*. Nodes forward packets on the basis of the information that they keep in a data structure called Forwarding Information Base (FIB). A FIB basically maps each network destination to a next-hop. In the presence of Equal Cost Multi-Path (ECMP), a single destination can have multiple load-balanced next-hops. We denote the set of next-hops in the FIB of any node r for an IP destination d at time t as $next(r, d, t)$. Observe that SDN-capable nodes can forward packets based on information different from the only destination IP address, e.g., the type of the traffic, as r_1 in Fig. 1. For SDN mechanisms, therefore, the concept of destination is more fine-grained (e.g., a destination could be the combination of IP address and port number).

Node populate their FIB relying on the information provided by *control-plane mechanisms*. A typical example of control-plane mechanism is a distributed routing protocol like OSPF, where each node computes routes to every destination. An even simpler example is static routing, in which each route is explicitly configured. Some newer mechanisms, like OpenFlow [5], can add and remove entries directly in the FIB of the nodes.

We classify control-plane mechanisms in Link State (LS), Distance Vector (DV), and SDN mechanisms. LS mechanisms are distributed protocols in which topological information is flooded to all the nodes, allowing each node to independently computes its FIB. OSPF and IS-IS are LS mechanisms. DV mechanisms are distributed protocols in which nodes do not have a global view of the network topology and compute their FIB relying on the information provided by their neighbors. RIP and EIGRP are examples of DV mechanisms. SDN mechanisms are protocols enabling a logically centralized SDN controller to modify the FIB of network nodes. Static routes and OpenFlow are SDN mechanisms.

2.2 Hybrid Networks

In hybrid networks, all the nodes run one IGP and one SDN mechanism. The SDN controller interacts with SDN-capable nodes using SDN specific protocols (e.g., OpenFlow), and with traditional nodes by configuring static routes. IGPs are also supported at SDN-capable nodes, e.g., as in [7]. We refer to LS hybrid networks or DV hybrid networks, depending on the IGP.

Two factors concur to select the mechanism that populates the FIB of each node for a specific destination d.

The first factor is the *mechanism preference*. It indicates the relative preference between different control-plane mechanisms. On IP routers, the preference between distributed protocols (including static routing) is controlled by a configuration parameter called Administrative Distance (AD). The AD can be set on a per-destination basis. OpenFlow is a special mechanism in that it accesses the FIB directly. For this reason, OpenFlow is effectively preferred over any other mechanism. Note that mechanism preference depends only on mechanism definitions and node configuration.

The second factor is *information availability* in each mechanism, i.e., the possibility for a given mechanism to provide any node with forwarding information to destination d at a given time t.

The combination of these two factors implies that, for each destination d, the FIB entry of node r at time t is populated by the r's most preferred mechanism with available information to reach d. We say that r *uses a mechanism* $m = used(r, d, t)$ for a destination d at time t if m is the source of r's FIB entries for d at time t.

Irrespectively of mechanism preferences, the following properties hold for different mechanisms.

PROPERTY 1. *Independently of control-plane mechanisms configured on any node in a network, for each destination, the next-hops of a node using a* DV *mechanism m are guaranteed to use m.*

PROPERTY 2. *For each destination, an* LS *mechanism provides each node with a non-empty set of next-hops despite the presence of other control-plane mechanisms in the network.*

PROPERTY 3. *For each destination, an* SDN *mechanism provides each node with a possibly empty set of next-hops despite the presence of other control-plane mechanisms in the network.*

Intuitively, Property 1 holds since, in a DV mechanism, a node propagates a routes to a destination d only if the route was used to compute its FIB entry to d. Moreover, Property 2 holds because LS mechanisms distribute topological information independently of the presence of other mechanisms. The same full visibility applies to SDN mechanisms, where the controller is supposed to known all the paths to each destination. However, the SDN controller can manipulate only a subset of FIB entries of each node, yielding Property 3.

2.3 The Safe Update Problem

To achieve manageability and evolvability in hybrid networks, we study the problem of disruption-free adaptation of forwarding paths in hybrid networks. We globally refer to the configuration and preference of mechanisms on nodes as network configuration, or simply *configuration*. Note that, in any configuration of a hybrid network, some flows are IGP-controlled and others are SDN-controlled. We denote a change from an

initial configuration to a final configuration as network update, or simply *update*. For the network update to be controllable, we consider stepwise update process (as in [12, 14]), where atomic operations are performed on a single node at each step.

As shown in Fig. 1, the operational order followed in the update can trigger path inconsistencies, i.e., forwarding anomalies only due to the network update process. We can state our problem as follows.

PROBLEM 1. *Given an initial and a final network configuration, compute an operational order to update the network without triggering any path inconsistency.*

We always assume that both the initial and the final configurations are anomaly-free. In the following, we always focus on forwarding paths that change between the initial and the final configurations. For the sake of simplicity, we assume that all the changing paths are controlled by an initial mechanism in the initial configuration, and by a final mechanism in the final configuration. However, all our findings and techniques equally apply to the most general case. The initial and final mechanism controlling some forwarding paths may be different (e.g., if IGP-controlled flows become SDN-controlled in the final configuration) or the same (e.g., if it runs different configurations).

The network update process consists in progressively changing the configuration of nodes until all the nodes in the network run the final configuration. The set of possible operations on a node is (i) modification of mechanism preference, (ii) addition or removal of static routes on traditional nodes, and (iii) addition or removal of FIB entries via SDN mechanisms on SDN-capable nodes. Each of those operations can be restricted to an arbitrary set of destinations. Unless differently specified, time is discrete and refer to stable forwarding states in each update step. Times $t = 0$ and $t = f$ stand for the initial and final configurations, respectively.

To define forwarding anomalies, we need to formally define forwarding path as concatenation of FIB entries. More precisely, a *forwarding path* $\pi(r, d, t)$ from r to d at time t is a sequence of nodes $(v_0 \dots v_k)$ such that $k \geq 0$, $v_0 = r$, and $\forall i = 0, \dots, k-1$, $v_{i+1} = next(v_i, d, t)$. Each node can appear at most once in $\pi(r, d, t)$.

A *path inconsistency* occurs during an update, at time t, if $\pi(r, d, t) \neq \pi(s, d, 0)$ and $\pi(r, d, t) \neq \pi(s, d, f)$, i.e., the forwarding path from r to d at t is a path different from both the forwarding paths in the initial and in the final configurations. Path inconsistencies force traffic to be forwarded through unpredicted forwarding paths, potentially disrupting routing policies (e.g., bypassing middle-boxes, as in Fig. 1) and traffic engineering (e.g., negatively affecting delay and making congestion harder to predict).

Connectivity problems during a network update are special cases of path inconsistencies, in which $\pi(r, d, t) = (r \dots v_k)$ and v_k has no direct connectivity to destination d. In particular, we distinguish between blackholes and forwarding loops. A *blackhole* occurs if $\pi(r, d, t) = (r \dots v_k)$ and $v_k \neq d$ and node v_k has no FIB entry for d. A *forwarding loop* occurs if $\pi(r, d, t)$ contains repeated nodes. Both blackholes and forwarding loops directly lead to packet dropping.

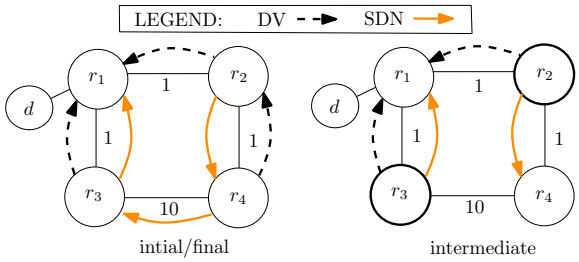


Figure 2: A case in which a blackhole can occur in a hybrid network update.

In the following, we assume that each mechanism is anomaly-free in the absence of other mechanisms. This means that, when taken in isolation, the used SDN mechanism provides connectivity between every source node and every SDN-controlled destination. The same always property holds in IGPs for all the destinations in the network by their definition.

3. ENSURING CONNECTIVITY

In this section, we show that **end-to-end connectivity can always be preserved** during network updates. We first describe how to avoid blackholes (§3.1), and then we focus on forwarding loops (§3.2)).

3.1 Blackholes can always be avoided

Fig. 2 shows a simple case in which a blackhole can be created during a hybrid network update. Assume that the initial mechanism controlling paths to destination d is a DV mechanism, while the final one is an SDN mechanism. Then, the update consists in reconfiguring each node, one node at a time, by replacing the DV information with the SDN one in its FIB entry to d. The forwarding paths in the initial configuration (black dashed arrows) are computed according to the IGP link weights. The other arrows denote the FIB entries installed in the final configuration. As evident in the left part of the figure, the initial and the final network configuration are anomaly-free. Consider now the case (shown in the right part of the figure) in which r_2 and r_3 are the first nodes to be reconfigured. Hence, at $t = 2$, r_2 and r_3 use the SDN mechanism. By Property 1, both r_2 and r_3 stop propagating to r_1 DV routes for d. Moreover, as long as SDN entries are installed only on reconfigured nodes, the SDN mechanism does not provide r_1 with forwarding information to reach d, creating a blackhole at r_1.

Intuitively, the blackhole is caused by the inability of both the initial and the final control-plane mechanisms to provide all the nodes with enough information to fill their FIBs. In the example in Fig. 2, the order in which nodes were updated prevented the the correct distribution of routes in the DV mechanism, while the SDN mechanism was not yet deployed on the affected node. Similar blackhole can also occur in updates of LS hybrid networks. For example, in the previous example, a blackhole can also occur during a update from an LS mechanism to an SDN one, if the support for the LS protocol is removed when a node is reconfigured.

Luckily, blackholes can be prevented by design of the update technique. Indeed, the technique can force the final mechanism to provide nodes with backup routes before starting the update. If the final mechanism is LS, each node can be guaranteed to have a FIB entry to each destination by activating the final mechanism on all the nodes before starting the network update, because of Property 2. Otherwise, static routes reflecting the final next-hops and pre-installed on nodes before starting the update, can be used as backup routes. In both cases, backup routes are initially installed as the least preferred mechanism. When a node is reconfigured, its backup routes are either promoted to more preferred (e.g., in the case of LS mechanisms), or removed and replaced by the information provided by the final mechanism (e.g., in the case of static routes).

In the following, we exclusively consider techniques that prevent blackholes by design.

3.2 Forwarding loops can always be avoided

Fig. 1 shows a case in which a forwarding loop occurs during a hybrid network update. We now show that forwarding loops can always be prevented.

First, consider the case of LS hybrid networks. Because of Property 2 and 3, the following holds.

PROPERTY 4. *Consider any update involving only* LS *and* SDN *mechanisms. At each step, for each destination d, each node uses its initial next-hops to d if not reconfigured, and its final next-hops to d otherwise.*

Property 4 has the important consequence that updating a node n only affects n's next-hops. Based on this, it is possible to compute operational orderings that prevent forwarding loops during LS IGP reconfigurations. In [14] it is shown that a per-destination ordering that avoids forwarding loops always exists. Since the technique to compute the ordering only relies on Property 4, we can use the same algorithms to prevent loops in all the updates involving only LS and SDN mechanisms, i.e., in any update of LS hybrid networks.

DV hybrid networks do not even impose ordering constraints. In fact, any ordering is guaranteed to be free from forwarding loops. This is due to Property 1 that ensures that DV nodes coordinate, and compute a consistent routing tree to each destination. In particular, the following lemma holds.

LEMMA 1. *If a node r uses a* DV *mechanism m to reach a destination d, then all the nodes in the actual path from r to d use m.*

PROOF. We now show that, given any time t, any node r and any destination d, if $used(r, d, t)$ is a DV mechanism m, then $\forall u \in \pi(r, d, t)$, $used(u, d, t) = m$. Let $\pi(r, d, t) = (r\ x)P$. If $x = d$, then P is empty, and the statement directly follows. Otherwise, x must use m by Property 1. The statement follows by iterating the same argument until the destination is reached. □

Based on Lemma 1, we can group all nodes that use the DV mechanism m to reach d at time t in a set $cone(m, d, t)$ which we call DV-*cone* . Lemma 1 states that the forwarding path of each node in the DV-cone is internal to the DV-cone itself. Hence, once a packet arrives at a node in the DV-cone of a destination d, the packet is guaranteed to reach d. As a consequence, the following theorem holds.

THEOREM 1. *No forwarding loop can occur during network updates involving a* DV *mechanism.*

PROOF. Let m_1 and m_2 be the two mechanisms involved in the update. Without loss of generality, assume that m_2 is DV. By Lemma 1, for each node r, destination d and time t, $\pi(r, d, t) = PQ$, where P is a possibly empty path containing only nodes using m_1 and Q is a possibly empty path including only nodes using m_2. Since no node can simultaneously use m_1 and m_2, $P \cap Q = \emptyset$. Since we assume anomaly-free mechanisms, both P and Q do not contain blackholes nor forwarding loops, yielding the statement. □

4. ENSURING PATH CONSISTENCY

Contrary to blackholes and forwarding loops, an operational ordering that avoids path inconsistencies may not exist. Consider, for instance, the scenario depicted in Fig. 3, in which d is the only destination. Assume a LS hybrid network update. Independently of which are the initial and final mechanisms controlling d, no node among u, v and z can be reconfigured without causing a path inconsistency. Indeed, by Property 4, each node uses its final next-hops if and only if reconfigured. Hence, reconfiguring v as the first one will create a forwarding loop between v and z. Moreover, starting the update from either u and z will create path inconsistencies on u. Similar cases exist in DV hybrid networks.

In the following, we propose an algorithm to avoid path inconsistencies (§4.1), called *Generic Path Inconsistency Avoider (GPIA)*. GPIA is correct and complete, in the sense that it is free from both false positive and false negatives. It applies to any update of

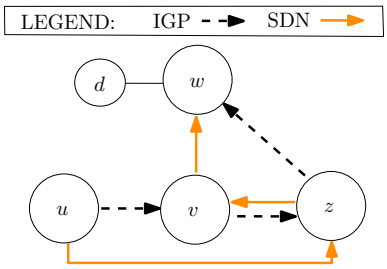


Figure 3: A case in which no operational ordering can prevent path inconsistencies in LS hybrid networks.

both LS and DV hybrid networks. However, the algorithm has different complexity different update cases (§4.2). Even when no reconfiguration ordering exists, GPIA returns a maximal sequence of nodes that can be reconfigured without creating path inconsistencies. We propose variations of the update technique to increase the number of nodes that can be safely reconfigured (§4.3), and fallback solutions to be applied to the remaining nodes (§4.4).

4.1 The GPIA algorithm

Intuitively, to check if the update of a single node causes a path inconsistency, we can simply simulate the change, computing the resulting forwarding paths, and comparing them with the initial and final ones. We use this intuition in the *check_consistency* procedure, illustrated in Fig. 4. Since forwarding paths can be easily built by concatenating next-hops used by each node, the critical step is to calculate next-hops, given a set of reconfigured nodes (sub-procedure *compute_next_hops*). By Property 4, any LS or SDN mechanism always provides each node with either the initial or the final next-hops (depending on whether the mechanism are the initial or the final one) to each destination. For a DV mechanism, computing next-hops at a given node implies calculating the shortest path in the DV-cone from that node to each considered destination. Then, the next-hops actually used by each node depend on its mechanism preferences, i.e., on whether the node is reconfigured or not. In any case, the *check_consistency* procedure takes polynomial time with respect to the number of nodes. Observe that, for the sake of simplicity, we deliberately omitted a number of possible performance optimizations. For instance, forwarding paths can be stored and re-used for many nodes. A detailed description of possible performance optimizations is outside the scope of this paper.

The *check_consistency* procedure can be used as a building block to compute a reconfiguration ordering that avoids path inconsistencies, as in the Generic Path Inconsistency Avoider (GPIA) algorithm. The algorithm, reported in Fig. 5, tries to iteratively compute

1: ***check_consistency***$(r,\ d,\ next_0,\ next_f,\ M)$
2: $nhs \leftarrow compute\_next\_hops(M \cup \{r\}, d)$
3: $curr\_paths \leftarrow build\_actual\_paths(nhs)$
4: **return** $curr\_paths = build\_actual\_paths(next_0) \vee curr\_paths = build\_actual\_paths(next_f)$

Figure 4: Procedure to check absence of path inconsistencies to d if a node r is reconfigured.

1: ***compute_sequence***$(N,next_0,next_f,D,M,max\_seq)$
2: $seq \leftarrow []$
3: **for** $d \in D$ **do**
4: $C_d \leftarrow \emptyset$
5: **for** $x \in N$ **do**
6: **if** check_consistency$(x, d, next_0, next_f, M)$ **then**
7: $C_d \leftarrow C_d \cup \{x\}$
8: **end if**
9: **end for**
10: **end for**
11: $C \leftarrow \cap_{d \in D} C_d$
12: **for** $n \in C$ **do**
13: $tail \leftarrow$ compute_sequence$(N \setminus \{n\}, next_0, next_f, D, M \cup \{n\})$
14: $seq \leftarrow$ concat$([n], tail)$
15: $max\_seq \leftarrow update\_max\_seq(max\_seq, seq)$
16: **if** $|seq| = |N|$ **then**
17: break
18: **end if**
19: **end for**
20: **return** max_seq

Figure 5: GPIA Algorithm.

a safe reconfiguration sequence. At each iteration, for each destination d, GPIA builds a set C_d of candidate nodes that can be reconfigured without generating a path inconsistency to d. Then, GPIA picks any node in the intersection of all sets C_d, and iterates until it finds an empty intersection. If the computed sequence includes all the network nodes, then it is directly returned. Otherwise the algorithm backtracks on previous candidate choices. Finally, GPIA returns the maximal sequence it found.

GPIA finds the maximal sequence of nodes that can be reconfigured without creating path inconsistencies. Indeed, by definition, the *check_consistency* procedure identifies all the nodes that can be safely reconfigured at a given update step. Exploration of all safe operational orderings is ensured by the GPIA backtracking ability.

4.2 Computational complexity of GPIA

The computational complexity of GPIA depends on the recursion tree generated in the backtracking phase. In the worst case, this recursion tree enumerates all possible orderings, yielding combinatorial complexity.

However, if updating a node in the candidate set C cannot remove any other nodes from C, only a single branch of the recursion tree needs to be explored. In this case, GPIA behaves as a greedy algorithm, and takes

polynomial time with respect to the number of nodes. The following theorem shows that this holds in network updates involving only LS and SDN mechanisms.

THEOREM 2. *Consider an update involving only* LS *or* SDN *mechanisms m_1 and m_2, and assume that node r is in set C at any time t. Then, r will stay in set C until it is reconfigured.*

PROOF. Assume by contradiction that node r is removed from set C at time $t+1$. Let x be the node reconfigured at time $t+1$. By definition of GPIA, $x \in C$ at time t. If x does not change next-hops, then a contradiction follows by Properties 2 and 3. Hence, it must be $next(x, d, 0) \neq next(x, d, f)$.

By construction of C, $r \notin C$ means that reconfiguring r at time $t+2$ creates a path inconsistency for at least one destination d. By Property 2 and 3, reconfiguring r can only change the next-hops of r, so we must have $next(r, d, 0) \neq next(r, d, f)$. Since $r \in C$ at time t, we have two cases:

- $r \in \pi(x, d, t+1)$. At time $t+1$, x has been reconfigured, i.e., $next(x, d, t+1) \neq next(x, d, 0)$, which implies $\pi(x, d, t+1) \neq \pi(x, d, 0)$. Moreover, r has not been reconfigured yet, i.e., $next(r, d, t+1) \neq next(r, d, f)$, which implies $\pi(x, d, t+1) \neq \pi(x, d, f)$.
- $x \in \pi(r, d, t+1)$. We have that $next(x, d, t+1) \neq next(x, d, 0)$ implies $\pi(r, d, t+1) \neq \pi(r, d, 0)$. On the other hand, $next(r, d, t+1) \neq next(r, d, f)$ implies $\pi(r, d, t+1) \neq \pi(r, d, f)$.

In the first and second cases, reconfiguring x at time $t+1$ triggers a path inconsistency at node x and r, respectively, hence contradicting that $x \in C$ at t. □

COROLLARY 1. *The time complexity of GPIA is $O(n)$, where n is the number of network nodes.*

In the presence of a DV mechanism, Property 1 forces nodes that use the DV mechanism to select a next-hop in the DV-cone. Since the shape of the DV-cone changes at each step, a node can exit the candidate set C as a consequence of the reconfiguration of another node. This implies that GPIA cannot behave as a greedy algorithm in the general case. However, a result similar to Theorem 2 holds for network updates that are path-preserving, i.e., in which $next(r, d, 0) = next(r, d, f)$ for each node r and each destination d.

THEOREM 3. *Consider a path-preserving update such that the initial mechanism m_1 is* DV *and the final mechanism m_2 is not. If a node r is in set C at any time t, then r will stay in set C until it is reconfigured.*

PROOF. Assume by contradiction that node r is removed from the candidate set C at time $t+1$. Let x, such that $x \in C$ at time t, be the node that GPIA reconfigures at time $t+1$. By construction of C, reconfiguring r at time $t+2$ creates a path inconsistency for some node u and some destination d. For the path inconsistency to occur, reconfiguring r must force u to change its forwarding path. Hence, by Property 1, we must have $r \in \pi(u, d, t+1)$.

Consider now the forwarding path of u at time t. If $r \in \pi(u, d, t)$, then reconfiguring r at time $t+1$ would cause a path inconsistency, as it occurs at time $t+2$ by definition of u. This contradicts the hypothesis that $r \in C$ at time t. Otherwise, $r \notin \pi(u, d, t)$ but $r \in \pi(u, d, t+1)$. This implies that reconfiguring x at time $t+1$ causes a path change at u. However, since the update is path-preserving, $\pi(u, d, 0) = \pi(u, d, f)$, hence each path change results in a path inconsistency. This contradicts the hypothesis that $x \in C$ at time t. □

Theorem 3 allows us to carry out arbitrary updates from a DV mechanism to SDN at the cost of doing them in two phases. In the first phase, we can perform a path-preserving update, which can be efficiently tackled with GPIA. In the second phase, we use again GPIA to change the forwarding paths in the SDN mechanism.

4.3 GPIA Variants

Intuitively, each destination poses its own set of constraints. Hence, the length of the maximal reconfiguration sequences (e.g., the ones computed by GPIA) generally decreases with the number of destinations. Based on this observation, we propose two variants of GPIA that improve its effectiveness in specific update cases.

GPIA with progressive destination removal. Thanks to the routing protocol definition, destinations can be safely removed from the initial mechanism if it is an IGP. Let d be a destination removed from the IGP at a given time. Indeed, as soon as the IGP converges, the old information is purged and all the nodes start using the final mechanism to forward packets to that destination. The correctness of the final configuration prevents permanent anomalies. Hence, the reconfiguration technique can progressively remove destinations when nodes are reconfigured. In case of progressive destination removal (PDR), reconfiguring a node affects its mechanism preferences and forces it to stop advertising connected destinations in the initial IGP, e.g., by means of `route-maps`. We refer to the corresponding variant of GPIA as PDR GPIA.

However, PDR also has two main drawbacks. The first one is that it cannot be applied if the initial mechanism is SDN, since it relies on dynamic information exchange between nodes. The second drawback is that removing a destination incurs a penalty in terms of convergence time and transient path inconsistencies. However, those anomalies are transient, and IGP convergence is normally fast. Moreover, techniques [15, 16]

are known that prevent blackholes and forwarding loops in most IGPs.

GPIA with destination distinction. Not all destinations are equally important in a network. For example, it is known that a relatively tiny number of traffic flows attract a disproportionate amount of traffic (see, e.g., [11]). GPIA can easily be adapted to consider different requirements for different destinations. In particular, the algorithm can receive as input only the subset of the destinations for which some form of path consistency is needed. For the input destinations, stricter or looser path consistency can be specified, e.g., to tolerate path inconsistencies for internal destinations as long as forwarding loops and blackholes are avoided. Such a logic is easily accommodated in GPIA when constructing the candidate set C_d.

4.4 Fallback Solutions

Fig. 3 shows that sometimes preventing path inconsistencies is simply impossible. In those cases, we rely on other technical solutions.

Static destination removal. We have already discussed that destinations can be safely removed from the initial mechanism if it is an IGP (see §4.3). Hence, in case no operational ordering exists on the nodes, destinations can be removed one by one by the initial mechanism. Each time a destination is removed from the IGP, all nodes will consistently use the final mechanism as soon as the initial IGP converges.

Static destination addition. When the final mechanism is an IGP, more specific destinations can be added to the final IGP in order to force its usage by all the nodes in the network in a single step. Consider again the example in Fig. 3. We can partition the destination prefix d in two sub-prefixes d' and d''. Then, we inject d' in the final IGP. At this step, all the nodes in the network will start consistently using the final IGP (hence the final forwarding paths) for traffic to d', because of the longest prefix matching rule. Moreover, all nodes keep using the initial mechanism for the rest of traffic to d since they are not reconfigured. Announcing d'' in the IGP has an analogous effect. After this second step, all nodes are using the final IGP for packets to any IP address in d. Hence, reconfiguring nodes in any order will not trigger any update anomaly.

Moreover, if d contains a single IP address, e.g., as for loopback interfaces, a new IP address can be added among those assigned to the destination. This new IP address can either be kept in the final configuration, or be used only during the update, switching back to the initial IP address as the last update step.

Destination addition and destination removal also provide a safe way to perform any update of hybrid networks at the cost of possibly introducing transient anomalies, adding management overhead and slowing down the update process. On one hand, transient anomalies during IGP convergence are highly mitigated by advanced techniques to prevent anomalies (e.g., transient loops [15, 16]) and to speed up IGP convergence (less than one second even in large networks [17]). On the other hand, however, the configuration of single nodes needs to be modified multiple times, i.e., to add or remove routes to given destinations and possibly temporary IP addresses to nodes. Moreover, destination addition would increase the size of the nodes' FIB only for the purpose of the update. For those reasons, we propose to use those solutions only if a reconfiguration ordering does not exist.

5. EVALUATION

In this section, we evaluate the effectiveness of our techniques, through simulations of realistic update cases in which forwarding paths are changed for traffic flows to one or several destinations. We consider LS and DV hybrid networks in Sec. 5.1 and 5.2, respectively. We stress that our techniques can be applied to further scenarios, like moving traffic flows away from specific links or nodes to avoid congestion.

5.1 Updates of LS hybrid networks

We started simulating network updates of LS hybrid networks in which IGP-controlled flows become SDN-controlled or vice versa. During such control shifts, the forwarding path followed by the affected flows changes as to mimic traffic engineering operations.

As dataset, we used the topologies computed and made publicly available in the Rocketfuel project. For each topology, we partitioned the nodes according to the cities in which they are located. We refer to nodes having no direct links with nodes in other cities as *border nodes*. Border nodes are 38 out of 306 nodes for AS 1221, 10/322 for AS 1755, 27/656 for AS 3257, 19/294 for AS 3967, and 73/748 for AS 6461. Since border nodes attract Internet traffic, we assumed them to be critical for both traffic engineering and policy routing requirements. In our experiments, we use border nodes as destinations of the traffic flows that we modify.

Single destination

As a first use case, we reconfigured traffic flows to one border destination. We make several experiments on each topology. In each experiment, we pick a border node as destination, we change some forwarding paths to the picked destination, and we run algorithm GPIA. We computed the old forwarding paths as shortest paths on the original IGP graph. The final shortest paths coincides with shortest paths on another weighted graph that differ from the IGP one for 10 link weights. This can reflect a case in which IGP- and SDN- forward-

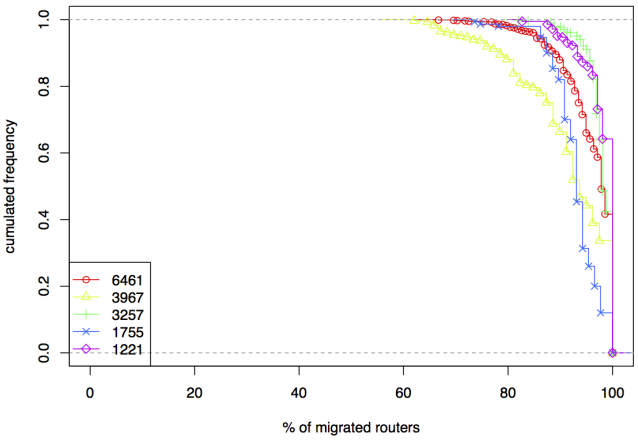


Figure 6: CCDF of nodes that can be safely reconfigured using GPIA in the single destination experiments on LS hybrid networks.

ing paths are computed according to different criteria, e.g., using bandwidth for IGP link weights, and delay as the main metric for SDN-controlled paths. This would support different forwarding path computation for different kinds of traffic, e.g., letting IGP manage traffic from standard applications while SDN supports VoIP services. The control shift of a flow from IGP to SDN (or vice versa) may be then needed depending on the relocation of end users and servers.

For each pair of topology and destination, we ran 15 simulations. In each simulation, we randomly select the link weight to be changed and we assign a new value to the link consistent with the values of other links.

Fig. 6 shows the Complementary CDF (CCDF) of the percentage of nodes that can be reconfigured without causing path inconsistencies. A point (x, y) in the plot indicates that at least $x\%$ of the nodes can be safely reconfigured in $y*100\%$ of the experiments. Despite a safe update sequence does not always exist, an update sequence that provably prevents any forwarding anomaly during the entire update can be applied in many of our experiments. In the remaining cases, the vast majority of nodes (typically, more than 80%) can be safely reconfigured, with few exceptions in which a safe ordering cannot be found for about 40% of the nodes. Those results imply that GPIA allows the less efficient fallback solutions proposed Section 4.4 to be applied to a small percentage of nodes.

Multiple destinations

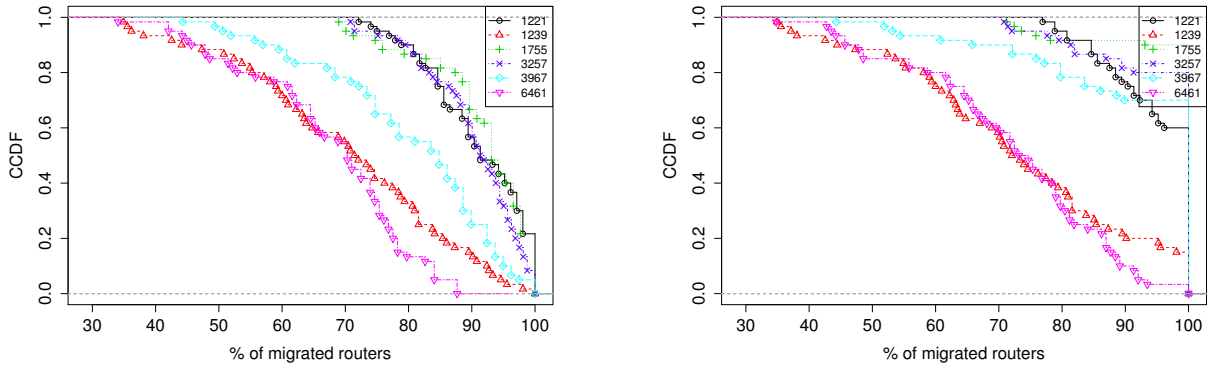
To react quicker to some network events, like increasing load threatening congestion, or for traffic engineering purposes, it is often desirable to change paths to multiple destinations at the same time. Moreover, simultaneously reconfiguring several destinations would minimize the time during which the network is in intermediate states, and forwarding paths are not optimized.

To assess the effectiveness of our techniques when traffic flows to multiple destinations are changed, we performed two series of experiments. As in the single destination experiments, forwarding paths were computed as shortest paths on different graphs, differing by 5 and 10 link weights in the first and second series of experiments, respectively. Observe that state of the art techniques for traffic engineering [18] are stated to typically few (not rarely, less than 10) link weights. For each topology and each series of experiments, we also run the PDR version of GPIA (see §4.3), imagining to apply it when the initial mechanism is an IGP. To perform a worst case analysis, we considered all the border nodes as destinations. For statistical significance, we ran each series of experiments 60 times, randomly selecting each time the links having different weights in the initial and final graphs.

The results for the 5 link changing experiments are reported, as CCDF, in Fig. 7. Fig. 7(a) shows that, for 3 topologies out of 6, the large majority of the nodes (more than 80% in more than 80% of the experiments) could be safely reconfigured following the ordering computed by our GPIA algorithm. In the remaining topologies, a lower but still significant percentage of nodes (more 50% except a small percentage of the experiments) could also be safely reconfigured in many experiments. As in the single destination experiments, running GPIA significantly limits the number of cases in which to apply solutions less efficient than a simple operational ordering.

Fig. 7(b) reports the results for the same experiments as obtained by running PDR GPIA. As predicted by the theory, removing destinations improves the effectiveness of our update technique, and enables GPIA to find longer node update sequences. Contrary to the original GPIA algorithm, the PDR version also finds a complete ordering in many cases, i.e., in more than 60% of the experiments for 4 out of 6 topologies. Moreover, in the same 4 topologies, more than 90% of the nodes can be safely reconfigured following our computed ordering in 70% of the experiments (and few less in the remaining experiments). Unfortunately, the improvements in the 2 other topologies are less important. This is not completely surprising as the effectiveness of GPIA is highly dependent on network topologies. We plan to have a deeper insight on the peculiarities of those topologies with respect to the others in future work.

By repeating the same experiments with 10 links, we found similar trends. In particular, on ASes 1221, 1755 and 3257, GPIA was able to safely reconfigure more than 60% of the nodes and more than 75% in about 80% of the experiments. ASes 6461 and 1239 are the ones on which GPIA performed the worst, though it was able to reconfigure about 60% of the nodes in more than 50% of



(a) original GPIA

(b) GPIA with destination removal

Figure 7: CCDF of nodes that can be safely reconfigured using GPIA in the multiple destination experiments on LS hybrid networks.

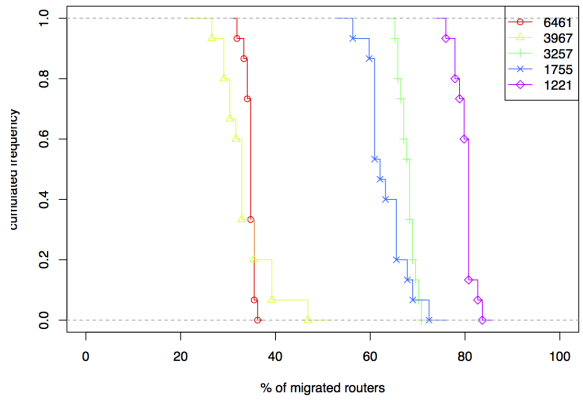


Figure 8: CCDF of nodes that can be safely reconfigured using GPIA in our experiments on DV hybrid networks.

the experiments. PDR GPIA improved those numbers similarly to the case of 5 link weight changes.

5.2 Updates of DV hybrid networks

We considered cases in which DV-controlled flows are reconfigured to become SDN-controlled in DV hybrid network. We focus on path-preserving update cases, i.e., in which the forwarding paths are the same in the initial and final configurations. This scenario can reflect a progressive transition of networks still deploying DV IGPs, as reported to be still frequent in enterprise networks [19]. Observe that the path-preserving case is the only one in which GPIA is polynomial when applied to DV hybrid networks (see §4.2), and it is also the more likely to have longer update sequences.

We performed experiments similar to those run for LS hybrid networks. Fig. 8 is the equivalent of Fig. 7(a) for DV to SDN updates to multiple destinations. We observe much less variability compared to Fig. 7(a), suggesting that changing 10 links only minimally affects the effectiveness of GPIA. Moreover, a complete sequence was impossible to find in any of the experiments, and the number of nodes that can be safely reconfigured were significantly lower than in the LS hybrid networks experiments. Contrary to LS hybrid networks, even augmenting GPIA with the destination removal feature provides marginal improvements (not shown in the figure for brevity). We ascribe such a discrepancy between LS and DV hybrid networks to Property 1 which imposes intermediate next-hops, depending on the shape of the DV-cone, to nodes in the DV-cone.

Our experiments show that forwarding can be hardly changed without disruptions in DV hybrid networks, even in path-preserving updates. Our results suggest to prefer LS hybrid networks on DV ones, unless only end-to-end connectivity has to be preserved in update operations.

6. DISCUSSION

Table 1 recaps our theoretical results. We now discuss their high-level implications and their generality.

Comparison between LS and DV hybrid networks. The guaranteed absence of forwarding loops in DV hybrid network should not be taken as an argument to prefer DV IGPs in hybrid networks. The presence of the DV mechanism significantly complicates the problem of avoiding path inconsistencies. As opposed to LS hybrid networks, GPIA does not have a polynomial time bound when DV mechanisms are involved. Efficient computation of the ordering can be done only in some scenarios (e.g., when the final mechanism is LS or SDN) at the cost of doubling the number of recon-

from \ to	SDN	LS IGP	DV IGP
SDN	forwarding loops † path inconsistencies *	forwarding loops † path inconsistencies *	no forwarding loops path inconsistencies $^{?}$
LS IGP	forwarding loops † path inconsistencies *	forwarding loops [14]† path inconsistencies *	no forwarding loops path inconsistencies $^{?}$
DV IGP	no forwarding loops path inconsistencies $^{\circledast}$	no forwarding loops path inconsistencies $^{\circledast}$	no forwarding loops path inconsistencies $^{?}$

†NP-hard, *P-time, $^{\circledast}$P-time if done in two steps,$^{?}$open problem

Table 1: Summary of our theoretical findings.

figuration steps. In fact, studying the computational complexity of avoiding path inconsistencies in DV hybrid networks remains an open research problem.

Types of control-plane mechanism. Our findings reveal that different types of mechanisms incur different types of anomalies. Moreover, SDN and LS mechanisms show very similar properties, while DV protocols are a special case. This suggests that, in order to study the update process, the most important feature of a control-plane mechanism is whether it computes the routes based on a partial (DV) or complete (LS and SDN) knowledge of the network topology.

Generality of our techniques. By pinpointing the key properties of mechanisms, our findings are general enough to cover additional use cases than the ones discussed in this paper. First, although we focus on hybrid enterprise networks, our results apply to hybrid data center networks as well, namely if some switches run distributed protocols like TRILL [20]. Moreover, our techniques directly apply to updates in pure SDN networks. Finally, they can be leveraged to replace one IGP with another in a traditional network in which routers run two IGPs at the same time.

Limitations. We studied one form of coexistence between SDN and IGPs. Indeed, except during network updates, we assume each flow to be either completely IGP-controlled or SDN-controlled. Alternative forms of cooperation, e.g., in which part of the forwarding paths is determined by an IGP and part by SDN [21], are also possible. Despite being more powerful, they complicate network management and troubleshooting. We leave the study of forwarding anomalies in different coexistence paradigms for future work. Finally, throughout the paper we assume that packets are not modified en route, e.g., by NAT or by OpenFlow rules. If hybrid networks are to offer a comparable flexibility to pure SDN networks, the impact of such advanced functions deserves further investigation.

7. RELATED WORK

To the best of our knowledge, this is the first paper proposing general disruption-free techniques for consistent update of hybrid networks. In [8], progressive introduction of SDN is reported in a running hybrid network. However, disruptions during update operations are not prevented in [8], e.g., when FIB entries are changed for traffic engineering purposes.

The rest of the related work targets either IGP reconfigurations or update of pure SDN networks.

IGP reconfigurations. Changing IGP configurations while avoiding disruptions has been the objective of several research efforts. However, previous work mainly focused on forwarding loops in link-state IGP reconfigurations. Notably, algorithms to avoid forwarding loops during IGP convergence in case of topological modifications have been proposed in [16]. In this paper, we prevent permanent loops triggered by inconsistencies due to reconfiguration techniques. A framework to minimize a given disruption function is proposed in [22], however this directly applies only to link weight changes. A technique to avoid forwarding loops during arbitrary configuration changes is described in [14]. We found that this technique can be reused to avoid loops in LS hybrid networks. Moreover, we developed more refined techniques to avoid all possible forwarding anomalies in both LS and DV hybrid networks. The relationship between our work and [14] is highlighted in Table 1.

SDN update. The work closer in spirit to ours is the consistent update of pure SDN networks presented in [12]. In that work, however, a pure SDN network has been assumed, and the same technique cannot be applied on hybrid networks where SDN protocols and IGPs interact during traffic flow changes. As they can also be applied to pure SDN networks, our techniques are more general. Moreover, they ensure per-packet consistent updates while avoiding packet modifications (e.g., tagging) and sensibly reducing resource consumptions with respect to [12]. Indeed, *our techniques duplicate FIB entries only when strictly necessary.* This is especially critical for SDN switches that typically support a limited number of flow entries.

Congestion-free techniques applicable in OpenFlow networks have also been recently proposed, e.g., [24]. We target to preserve accommodation of more critical requirements, like security policies, in hybrid networks By preventing any path inconsistency, our techniques provably preserves accommodation of higher-level requirements like security policies, while limiting the likelihood of congestion during the network reconfiguration.

8. CONCLUSIONS

SDN promises incomparable flexibility. To create incentives to bootstrap an incremental transition towards SDN, part of this flexibility should be made available in hybrid SDN networks, mixing SDN-capable devices and devices not supporting SDN protocols yet.

In this paper, we developed the machinery to realize anomaly-free updates of hybrid networks. Our techniques enable dynamic traffic engineering and policy routing at a fine-grained level, e.g., supporting conditional load balancing (e.g., addition of egress points to a given set of network flows if traffic becomes higher than a given threshold), on-the-fly addition and removal of network resources (e.g., making the traffic traverse the smallest possible subset of devices in a given set), or conditional selection of forwarding paths (e.g., selection of the forwarding path for specific flows on the basis of the delay experienced over that path).

Although this paper is targeted to hybrid enterprise networks, the same machinery can also be exploited in hybrid data center networks, in pure SDN networks and in traditional IGP networks. A full evaluation of those use cases is left for future work.

9. REFERENCES

[1] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" in *OSDI*, Oct. 2010.

[2] D. Erickson, G. Gibb, B. Heller, D. Underhill, J. Naous, G. Appenzeller, G. Parulkar, N. McKeown, M. Rosenblum, M. Lam *et al.*, "A demonstration of virtual machine mobility in an OpenFlow network," *Proceedings of ACM SIGCOMM (Demo)*, 2008.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1270–1283, 2009.

[4] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Hot-ICE*, 2011.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] Infonetics, "SDN, 40G/100G, and MPLS Control Plane Strategies: Global Service Provider Survey," 2012, white paper.

[7] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *HotSDN*, 2012.

[8] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN," in *SIGCOMM*, 2013.

[9] Z. Houidi and M. Meulle, "A new VPN routing approach for large scale networks," in *ICNP*, 2010.

[10] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable Ethernet for data centers," in *CoNext*, 2012.

[11] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger, "A methodology for studying persistency aspects of internet flows," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 23–36, Apr. 2005.

[12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM 2012*. ACM, 2012, pp. 323–334.

[13] L. Vanbever, S. Vissicchio, L. Cittadini, and O. Bonaventure, "When the cure is worse than the disease: the impact of graceful igp operations on bgp," in *INFOCOM*, 2013.

[14] L. Vanbever, S. Vissicchio, C. Pelsser, P. François, and O. Bonaventure, "Lossless migrations of link-state igps," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1842–1855, 2012.

[15] J. Garcia-Lunes-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Trans. Netw.*, vol. 1, no. 1, pp. 130 –141, feb 1993.

[16] P. Francois and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *Trans. on Netw.*, vol. 15, no. 6, pp. 1280–1932, 2007.

[17] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," *Comput. Commun. Rev.*, vol. 35, no. 3, pp. 33–44, 2005.

[18] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *Proc. INFOCOM*, 2000.

[19] D. A. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjálmtýsson, and A. Greenberg, "Routing design in operational networks: a look from the inside," in *Proc. SIGCOMM*, 2004.

[20] J. Touch and R. Perlman, "Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement," RFC 5556, May 2009.

[21] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," 2013.

[22] S. Raza, Y. Zhu, and C.-N. Chuah, "Graceful Network State Migrations," *Trans. on Netw.*, vol. 19, no. 4, pp. 1097–1110, 2011.

[23] N. P. Katta, J. Rexford, and D. Walker, ""Incremental consistent updates"," in *HotSDN*, 2013, to appear.

[24] H. Liu, X. Wu, M. Zhang, L.Yuan, R. Wattenhofer, and D. Maltz, "zUpdate: Updating Data Center Networks with Zero Loss," in *SIGCOMM*, 2013, to appear.