Safe Update of Hybrid SDN Networks

Stefano Vissicchio, Laurent Vanbever, Luca Cittadini, Geoffrey G. Xie, and Olivier Bonaventure

***Abstract*— The support for safe network updates, i.e., live modification of device behavior without service disruption, is a critical primitive for current and future networks. Several techniques have been proposed by previous works to implement such a primitive. Unfortunately, existing techniques are not generally applicable to any network architecture, and typically require high overhead (e.g., additional memory) to guarantee strong consistency (i.e., traversal of either initial or final paths, but never a mix of them) during the update. In this paper, we deeply study the problem of computing operational sequences to safely and quickly update arbitrary networks. We characterize cases, for which this computation is easy, and revisit previous algorithmic contributions in the new light of our theoretical findings. We also propose and thoroughly evaluate a generic sequence-computation approach, based on two new algorithms that we combine to overcome limitations of prior proposals. Our approach always finds an operational sequence that provably guarantees strong consistency throughout the update, with very limited overhead. Moreover, it can be applied to update networks running *any* combination of centralized and distributed control-planes, including different families of IGPs, OpenFlow or other SDN protocols, and hybrid SDN networks. Our approach therefore supports a large set of use cases, ranging from traffic engineering in IGP-only or SDN-only networks to incremental SDN roll-out and advanced requirements (e.g., per-flow path selection or dynamic network function virtualization) in partial SDN deployments.**

***Index Terms*— Network management, network updates, reconfiguration, hybrid SDN, theory, algorithms, simulations.**

I. Introduction and Related Work

NETWORKS have to be updated often, e.g., to support prompt failure reaction, traffic engineering, policy changes and service deployment [1]–[3]. Network updates consist in modifying the forwarding table (Forwarding Information Base or FIB) of network nodes. An update can indeed be abstracted as a process that replaces initial FIB entries with final ones on all nodes. For reliability, updates should be performed incrementally, passing through a sequence of intermediate states. To avoid disruptions, consistency properties, like the absence of forwarding anomalies in every intermediate state or short update time, should be guaranteed. Many techniques have been proposed to carry out safe updates in various settings – see, e.g., [4] for a survey.

Unfortunately, the applicability of mainstream approaches is limited for two reasons: (i) the overhead that they impose to guarantee strong consistency during the update; and (ii) their lack of generality. Table I details those limitations.

Regarding overhead, prior techniques can be classified in two families. The first family (e.g., [2], [5]–[7]) guarantees strong consistency by consuming extra network resources. Those techniques either duplicate FIBs to install both the initial and final entries at the same time on all nodes [2], [5], [6], or temporarily store packets during updates [7]. Since network resources are typically limited and expensive (e.g., FIBs of OpenFlow switches implemented with TCAMs), those techniques are not always practical. The second family (e.g., [8]–[15]) is based on replacing initial FIB entries with final ones in a carefully-computed order, which induces no update overhead. Most of those techniques focus on weaker guarantees than strong consistency, e.g., absence of forwarding loops [8], [11], all connectivity disruptions [12], [13], congestion [9], [10] or middlebox bypassing [14]. Moreover, the few techniques (e.g., [15]) that support strong consistency have intrinsic limitations since a FIB-entry replacement order that guarantees strong consistency does not exist in some cases.

Regarding generality, prior techniques can only be applied to networks running either (i) solely a centralized control-plane (e.g., OpenFlow [2], [3], [5]–[7], [9]–[11], [15]), or (ii) exclusively a distributed one (e.g., a link-state IGP [8], [16], [17]). Hence, they do not support a wide set of use cases, including (i) incremental SDN roll-out; (ii) protocol replacement, e.g., of OpenFlow with future SDN protocols; and (iii) update of hybrid SDN networks [18], running both traditional control-plane protocols (IGP, BGP, etc.) and an SDN controller. We expect hybrid SDN to become the norm rather an exception in the future, as also suggested by documented SDN deployments [1], [9]: This likely makes those use cases more and more important over time.

Overcoming limitations of previous techniques means computing quick updates of generic networks, possibly running multiple control-planes, while guaranteeing strong consistency (also called *safety* in the following) with low overhead. This is hard for many reasons. First, guaranteeing strong consistency during an update is challenging. Indeed, we need to compute an operational sequence ensuring that no traffic flow is forwarded on any combination of initial and final paths, in each transient state of the update. Second, this challenge is exacerbated by our additional requirements, that is, quickness,

Manuscript received November 25, 2015; revised September 25, 2016; accepted December 7, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Chen. Date of publication January 18, 2017; date of current version June 14, 2017.

S. Vissicchio is with the University College London, London WC1E 6BT, U.K., and also with the Belgian fund for scientific research, Université catholique de Louvain, Louvain-la-Neuve 1348 Belgium (e-mail: s.vissicchio@cs.ucl.ac.uk).

L. Vanbever is with ETH Zurich, 8092 Zurich, Switzerland (e-mail: lvanbever@ethz.ch).

L. Cittadini is with the Roma Tre University, 00146 Rome, Italy (e-mail: ratm@dia.uniroma3.it).

G. Xie is with the Naval Postgraduate School, Monterey, CA 93943 USA (e-mail: xie@nps.edu).

O. Bonaventure is with Université catholique de Louvain, Louvain-la-Neuve 1348 Belgium (e-mail: olivier.bonaventure@uclouvain.be).

Digital Object Identifier 10.1109/TNET.2016.2642586

TABLE I
OUR ALGORITHMS OVERCOME LIMITATIONS OF PRIOR TECHNIQUES (SHOWN IN CHRONOLOGICAL ORDER)

update technique	*feasible update*	*strong consistency*	*update overhead*	*supported control-planes*
Update protocol [16]	always	no	none	modified link-state IGP
Shadow config [17]	always	yes	FIB duplication	distributed protocols
Ordered IGP changes [8]	not guaranteed	no	none	link-state IGP
Two-phase commit [2], [5], [6]	always	yes	FIB duplication	OpenFlow
Packet storage [7]	always	no	packet storage	modified OpenFlow
Ordered replacements [9]-[14]	not guaranteed	no	none	OpenFlow
Update synthesis [15]	not guaranteed	yes	none	OpenFlow
Post-ordering duplication (this paper)	***always***	***yes***	***few FIB entries***	***any combination***

low resource consumption, and general applicability. These requirements transform the decision problem of finding a safe update sequence into an optimization problem where all safe operational sequences have to be (implicitly) compared. Even worse, these requirements cannot be easily accommodated simultaneously. For example, updating destinations one by one limits memory overhead (by adding at most one FIB entry per node at each step), but drastically increases the update time. Finally, the need for generality prevents us from relying on specific features of given control-planes, like the ability of OpenFlow switches to send packets to the controller. In contrast, it requires to consider control-planes with qualitatively-different behaviors under a common framework.

In this paper, we describe how we tackled those challenges. We present a deep problem analysis, leveraging a generic control-plane model. We analyze the consequent theoretical findings and their impact on the literature. From our theory, we also derive a procedure that efficiently computes quick, safe, low-overhead updates of networks running any combination of centralized and distributed control-planes. Our procedure combines FIB-entry replacement and duplication, availing of their strengths and limiting their drawbacks (see Table I). This way, it generalizes principles of the recent FLIP algorithm [19] to arbitrary networks beyond OpenFlow-only ones. Our procedure works in two steps: it first orders FIB-entry replacements until safe, and then uses minimal extra memory to complete updates. In both phases, it parallelizes operations to limit the network-update time.

This paper therefore makes the following contributions.

Compelling use cases, not supported by prior works (§II): We aim to enable management abstractions (similar to [2]) in hybrid SDN networks. This would provide support for use cases like safe and incremental roll-out of SDN, advanced traffic engineering in partial deployments, and dynamic management of virtualized functions in (partial) SDN networks.

Problem analysis (§III): We abstract the update problem posed by our use cases, and illustrate the limitations of algorithms that exclusively rely on FIB entry replacement or solely on FIB duplication. We also detail the impact of qualitatively-different control-planes on previous works.

Theory (§IV): We zoom into the problem of finding a safe, quick, zero-overhead update sequence, and relate its complexity to fundamental properties of run control-planes. Our findings shed new light on the current literature. We indeed show that a simple (greedy) strategy is optimal to ensure strong consistency whenever the involved control-planes do not react to FIB changes, like basic OpenFlow controllers and link-state IGPs. That is, computationally-hard algorithms (e.g., [15]) can be avoided for those use cases. Further, we characterize instances of the zero-overhead update problem that require structurally different algorithms with respect to existing ones.

Algorithms (§V): To support generic-network updates, we design new algorithms implementing the two steps in our procedure. In the first step, the GPIA (Generic Path Inconsistency Avoider) algorithm computes the longest sequence of parallel per-node FIB replacements that guarantees strong consistency with no overhead. Since this sequence may not complete the update, the second step runs another original algorithm, called FED (FIB Entry Duplicator), that judiciously duplicates FIB entries. We formally prove the correctness of our algorithms and two-phase procedure for any (hybrid SDN) network.

Evaluation and comparison with prior work (§VI): We simulate our algorithms in realistic scenarios mimicking incremental SDN deployment and traffic engineering cases. In most experiments, GPIA safely orders many FIB entry replacements (often for more than 70% of the nodes). By complementing GPIA with FED, we achieve quick, safe updates, either improving the update success rate or obtaining $80 - 100\%$ overhead reduction with respect to mainstream alternatives.

II. HYBRID SDN UPDATE USE CASES

Previous works (e.g., [1], [18], [20]) have shown that hybrid SDN networks are profitable. In particular, they allow operators to keep offered services (e.g., MPLS VPNs) and interaction with other networks (e.g., via BGP), while achieving some SDN advantages (improved manageability and flexibility) without a complete replacement of traditional network components (devices, protocols, etc.).

In this section, we show the practical importance of supporting fast and safe updates in hybrid SDN networks, through a set of use cases. Note that no previous work is directly applicable to these use cases. Indeed, existing techniques focus on either IGP-only or SDN-only networks (see Table I).

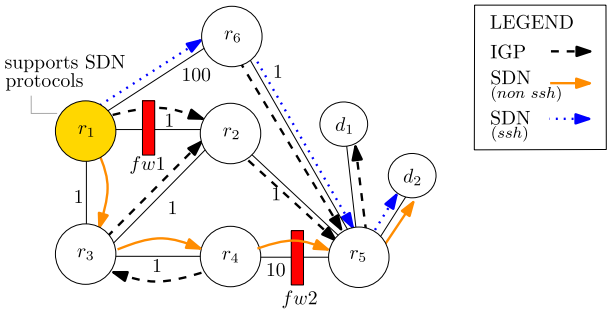


Fig. 1. Example of hybrid SDN network.

We illustrate the identified use cases on the network shown in Fig. 1, where a link-state IGP and an SDN controller coexist. Nodes (i.e., routers or switches) are represented by circles, and physical links by edges labeled with their IGP weights (that determine IGP paths through shortest-path computation). The only node also supporting SDN protocols (e.g., OpenFlow) is r_1, represented as a filled circle. The SDN controller can program the forwarding of all nodes including the IGP-only ones, e.g., by installing static routes, using standard interfaces to the routing system [21], or injecting information in the IGP [22]. The network contains two firewalls $fw1$ and $fw2$ (filled rectangles). In this example, traffic flows for destination d_1 (dashed arrows) are forwarded on IGP shortest paths. In contrast, the SDN controller imposes that flows for d_2 follow different paths depending on whether they carry `ssh` traffic (dotted arrows) or not (solid arrows).

Incremental deployment of SDN: Consider the case of an operator willing to introduce an SDN controller in an IGP-only network. To ensure a smooth and controlled transition and acquire confidence with the new paradigm, she may want to initially rely on the SDN controller only for a small portion of non-critical traffic. Fig. 1 can represent this first configuration. From there, the operator may want to roll back to the initial state if she has to debug the SDN controller, e.g., for unexpected behavior. Alternatively, after a trial period and maybe the deployment of additional SDN nodes, she may want to progressively move more and more flows (e.g., the ones from r_1 to d_1 in Fig. 1) under SDN control.

Both moving to a new network state and rolling back to previous ones require network updates in which flows are moved from IGP to SDN control, and vice versa.

Dynamic traffic engineering: In Fig. 1, only the flows for d_2 are controlled by SDN. This may be motivated by the need to adjust forwarding paths for the typically-few most important flows, e.g., deviating from shortest-path routing [20], [22]. For example, flows from r_1 to d_2 may carry high traffic volumes, or be subject to special requirements (e.g., low latency).

In those setups, network updates are needed whenever traffic distributions or requirements change. As an example, should a flash crowd occur for destination d_1, the SDN controller may have to tweak the forwarding of flows for d_1 and split them on different paths, possibly releasing the control of flows for d_2 if their volumes become negligible. Similarly, if a customer sending traffic from r_3 signs up for a low-latency service, the SDN controller should also take in charge the flows from r_3.

Dynamic and conditional SDN functionalities: Hybrid SDN networks can support SDN functionalities without requiring a full SDN deployment. For instance, in Fig. 1, fine-grained policies are applied even if r_1 is the only SDN-enabled node. Indeed, traffic from r_1 to d_2 is forwarded on different paths on the basis of its type. This can support security policies through middleboxing (e.g., forcing the non-ssh traffic from r_1 through a firewall), while maximizing the utilization of the network (e.g., load-balancing traffic from r_1). Similarly, virtualized network functions can be deployed on SDN nodes, and be used to process any flow traversing those nodes.

Network updates (of both IGP- and SDN-controlled flows) are needed to support SDN functionalities dynamically, e.g., depending on traffic volumes, or conditionally, e.g., under given network conditions. For example, the instantiation of virtualized security functions may be needed upon alarms (e.g., anomalous traffic surges) raised for specific flows; also, fine-grained forwarding policies may depend on the load of single middleboxes [23]. In Fig. 1, upon reception of suspicious traffic from r_3 and r_4, several forwarding paths may need to be updated, to enforce stricter security policies while avoiding link congestion. For example, the operator may want to (i) reconfigure the IGP network so that all r_4's flows traverse $fw2$; (ii) bring $r_3 - d_1$ flows under SDN control to force the corresponding traffic through $fw2$; and (iii) force $r_1 - d_2$ flows through $fw1$, to balance the load between $fw1$ and $fw2$.

In this paper, we treat **IGP-only and pure SDN networks as special cases of hybrid SDN ones**. Hence, our findings hold and our algorithms can be used as-is in IGP-only and SDN-only use cases studied by previous works, e.g., protocol replacements, traffic engineering in IGP or SDN networks, and dynamic functions in SDN-only networks.

III. Problem Analysis

To support the use cases presented in §II, we need the ability to safely and quickly update arbitrary networks at runtime.

An update can be generally defined as a *sequence of steps*, each including a set of operations run in parallel. The sequence replaces initial forwarding paths with final ones, by progressively changing nodes' FIB entries. We refer to any set of paths installed after the application of a sub-sequence of steps as intermediate paths or *intermediate state*.

To be safe, an update sequence must provably avoid intermediate states with *path inconsistencies*. A path inconsistency is a non-empty set of traffic-traversed paths that are different from both the initial and the final ones. Avoiding path inconsistencies ensures preservation of both forwarding correctness (delivery of packets to their destinations) and policies, e.g., security (traversal of a firewall, as for flows sourced at r_1 in Fig. 1) and performance (low-latency for certain flows) ones.

Since simultaneity of operations is impossible to guarantee in practice, we consider sequences as safe only if path inconsistencies are prevented independently of the relative order in which operations in the same step are executed by nodes. This condition indeed ensures strong consistency even if operations are delayed or not performed by nodes, e.g., because of lost

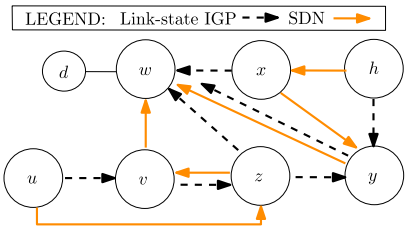


Fig. 2. An example in which no FIB replacement ordering can prevent path inconsistencies. The only applicable approach among prior works is network-wide FIB duplication [2], [5], [17], that comes with high overhead.

controller commands. However, it excludes naive approaches like executing all the operations in a single step.

A. Prior Techniques Cannot Achieve Safety With Low Overhead

Duplicating FIBs is not always practical: A straightforward approach to guarantee the absence of path inconsistencies is to maintain both the initial and final states on the nodes, i.e., *duplicating* their FIB. Packets can then be tagged, so that all the nodes process them consistently, according to packet tags. This approach is adopted in several update techniques, tailored to either SDN [2] or traditional [17] networks.

Duplicating the FIB on all nodes is generally unnecessary and not practical. Consider Fig. 2, where d represents a destination, and IGP paths for d have to be replaced with the SDN ones. Assume that y's FIB is almost full, with space for only 10 additional entries. Let $1,000$ destinations be attached to w as d, and the update represented in the figure affect all of them. In this case, we simply cannot duplicate y's FIB to perform the update, because there is not enough room in y's FIB. A workaround consists in splitting the update in multiple rounds [5], each updating 10 destinations. This, however, will significantly slow down the update, e.g., requiring to perform 100 sub-update procedures in our example, only to comply with FIB entry scarcity at some nodes (i.e., y).

Ordering FIB entry replacements is not always possible: Computing a safe sequence of FIB entry replacements, e.g., as proposed in[8]–[10] and [15], is the most effective approach to limit memory overhead. It consists in *replacing* (some) initial FIB entries of some nodes with the corresponding final ones, following a carefully-computed order.

Unfortunately, a safe update cannot be performed by only ordering FIB-entry replacements, in some cases. Consider again Fig. 2, with IGP paths to be replaced with SDN ones. In this case, FIB-entry replacements on u, v and z cannot be ordered without causing a path inconsistency. Indeed, starting from z creates an intermediate path including a forwarding loop between v and z. Also, starting the update from u or v leads to path inconsistencies at u, namely, $(u\ z\ w\ d)$ in the former case and $(u\ v\ w\ d)$ in the latter. In both cases, the $(v\ z)$ link is not traversed, potentially leading to a security violation (e.g., if a firewall is positioned on that link).

B. Prior Techniques Can Hardly Be Generalized

Different control-planes must be treated differently, depending on whether they react to node FIB changes or not [13].

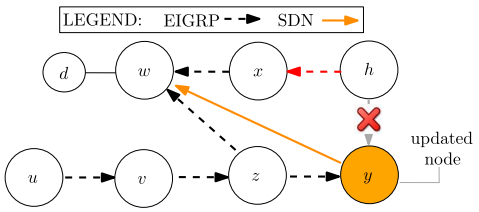


Fig. 3. EIGRP [24] triggers a path inconsistency *remotely* (on h), when updating one node (y). This path inconsistency is not avoided by previous algorithms, since they assume that FIB changes have only *local* effects.

Link-state IGPs and simple OpenFlow controllers (that we considered in the previous examples) do not react to FIB changes. However, protocols such as EIGRP (which is often used in enterprise networks [24]) react to FIB changes by withdrawing routes that traverse nodes whose FIB entries are not installed by the protocol itself. Similarly, SDN protocols like OpenFlow offer primitives for building controllers that are notified and react to FIB changes [13].

Reaction to FIB changes has an impact on the computation of safe updates. Assume that the IGP used in Fig. 3 is EIGRP. Let y be the first node to be updated, as shown in Fig. 3. As soon as y installs its final FIB entry, the IGP control-plane withdraws all the IGP paths traversing y (since y stops propagating its EIGRP routes to its neighbors). Consequently, h changes its FIB entry (even if not updated): It replaces its initial path $(h\ y\ w\ d)$ with the new shortest path for d including non-updated IGP nodes only. Should such new path be $(h\ x\ w\ d)$, we would have a path inconsistency.

The described path inconsistency is not possible with control-planes that do not react to FIB changes (e.g., link-state IGPs). In these cases, updating a node does not trigger remote effects. All prior update techniques assume the latter control-plane behavior. Hence, they do not support update consistency in the presence of more than one IGPs or multiple SDN controllers that react to FIB changes.

IV. Characterization of Update Complexity

In this section, we describe our model for generic hybrid-SDN networks (§IV-A), which captures key features of run control-planes abstracting from implementation details of specific protocols or controllers. We use this model to distinguish qualitatively-different updates (§IV-B). In particular, we characterize update scenarios for which it is easy to compute (e.g., with a greedy algorithm) a safe update sequence that does not duplicate FIB entries. The characterization is based on the nature of control-planes run before (i.e., initial) and after (i.e., final) the update, according to our model. Proofs of formal statements are reported in the Appendix.

We also analyze the consequences of our findings on previous works (§IV-C). Interestingly, prior techniques can be extended to hybrid SDN networks that are easy to update, while harder update cases require structurally-different approaches.

A. Modeling Hybrid SDN Networks

First, we describe how we model generic network, where we cannot make any assumption on which control-planes are run and how they compute nodes' FIBs.

We rely on a generic control-plane model: We adopt the model proposed in [13] that applies to networks with arbitrary combinations of uncoordinated control-planes, each running a different protocol (or a separate protocol instance). This model abstracts any control-plane as a process which possibly reads information from nodes, decides per-node forwarding entries, and installs them on the nodes. A control-plane is called FIB-aware (FA) if it computes forwarding entries on the basis of nodes' FIB content, as for typical implementations of distance-vector IGPs or SDN controllers that react to FIB changes. Otherwise, it is FIB-unaware (FU), as for link-state IGPs or simple SDN controllers.

By definition of FA and FU, the following properties hold.

Property 1: The FIB entries provided by an FU control-plane to any node are independent of the presence of any other coexisting control-plane.

Property 2: If the FIB entry of a node for a destination d is provided by an FA control-plane, the FIB entries of the successors of that node on any forwarding path for d are provided by the same FA control-plane.

We refer the reader to [13] for examples of FU and FA control-planes (including real device configurations) and for an in-depth discussion of their interactions.

In the following, we indicate network updates as *FU-only* if the FIB entries of all nodes to be updated are provided by FU control-planes before, after and throughout the update. *FA-involving* updates encompass all the other cases.

We support hybridly-controlled flows: Indeed, we admit that flows are forwarded over paths such that some subpaths are determined by one control-plane (e.g., an IGP) while other subpaths are provided by another control-plane (e.g., an SDN controller), as in [20]. For simplicity, we assume in the following that every flow is entirely controlled by a single control-plane in both the initial and final states – but obviously any flow can be controlled by a control-plane in the initial state and another control-plane in the final state. This assumption is *without loss of generality*, as detailed in the appendix.

B. Distinguishing Easy and Harder Updates

In [13], it has been shown that FU and FA are key to characterize anomalies due to coexisting control-planes.

We now show that those properties also determine the complexity of computing fast and safe update sequences with no overhead. We study how to compute sequence of *node updates*. A node update consists in replacing all the initial FIB entries of a node with its final ones. The following theorem stresses a fundamental difference between updates of different hybrid SDN networks, as it solely applies to FU-only updates.

Theorem 1: In an FU-only update, if a node r can be safely updated at time t, r can also be safely updated at any $t' > t$.

A greedy strategy guarantees fast and safe FU*-only updates:* Consider the greedy algorithm consisting in updating in parallel (i.e., in the same step) all the nodes that can be safely updated at a given time t. Theorem 1 implies that the resulting sequences guarantee strong consistency. The following theorem proves that such an approach is also optimal.

Theorem 2: For FU-only updates, the greedy strategy computes safe sequences of minimal length.

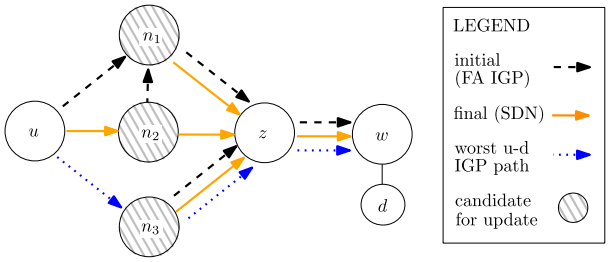


Fig. 4. A naive approach cannot be used for FA-involving updates. In this case, any among n_1, n_2, and n_3 can be safely updated, but updating all of them creates a path inconsistency at u (that will remain with no path for d).

Theorems 1 and 2 hold for any FU-only network. Further, they apply to more complex scenarios, like updates that do not change FIB entries on any FA-controlled device.

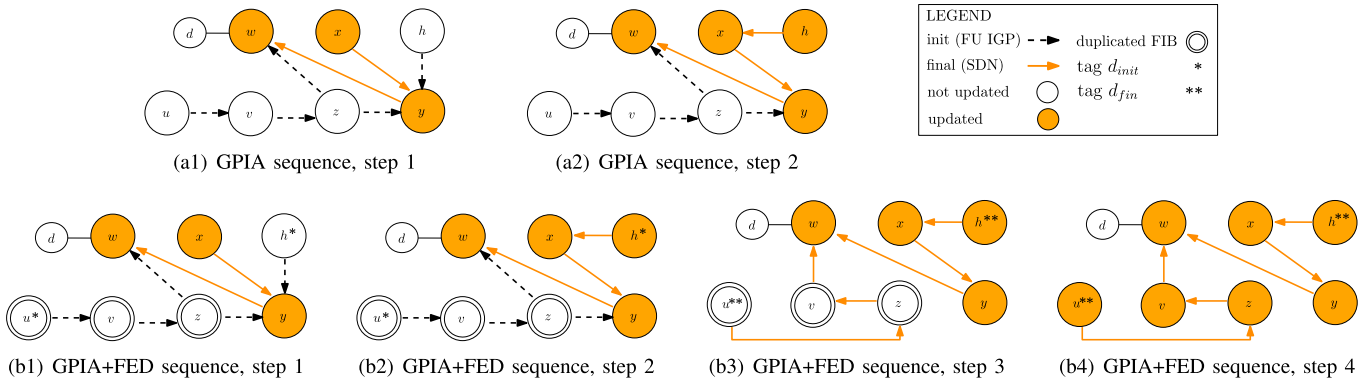
Updates involving FA *control-planes are harder:* Theorem 1 cannot be generalized to updates involving FA control-planes. Fig. 4 illustrates a counter-example. At time $t = 0$, updating u would trigger path inconsistency $(u\ n_2\ n_1\ z\ w\ d)$; similarly, updating z would withdraw all FA paths, leaving the non-updated nodes u, n_1, n_2 and n_3 without paths. In contrast, updating n_2 or n_3 is safe, since they are not used as next-hop by any other node in the initial state. Moreover, n_1 can be safely updated, because the second best paths in the FA IGP coincide with the final (SDN) paths. Nevertheless, updating n_1 would imply that n_2 cannot be updated anymore, contrary to what Theorem 1 guarantees for FU-only updates. Indeed, as soon as n_1 and n_2 are updated, u receives IGP paths only from n_3, hence it will start using path $(u\ n_3\ z\ w\ d)$, generating a path inconsistency. This also implies that an algorithm updating all n_1, n_2 and n_3 at the same time is not safe. Indeed, if the update on n_3 is slower than the other two, the path inconsistency $(u\ n_3\ z\ w\ d)$ will be triggered. Even worse, after the three nodes are updated, u would be left without any path for d (since it is not updated yet).

C. Impact on Previous Algorithms

Previous update algorithms proposed to compute safe FIB replacements either apply to classic OpenFlow or to existing link-state IGP protocols, that is, FU-only scenarios. Therefore, our findings come with the following consequences.

Previous algorithms can be used interchangeably in FU*-only networks:* In other words, algorithms targeting link-state IGP updates (e.g., [8]) can be translated into counter-parts for simple SDN controllers. Similarly, proposals for OpenFlow networks (e.g., [15]) can also be used in traditional networks (unless there is a mismatch between operation granularity). Finally, all previous approaches can be applied to hybrid SDN networks where control-planes do not react to FIB changes.

Previous algorithms may be unnecessarily complex for strongly-consistent FU *updates:* Theorem 1 implies that a greedy strategy preserves strong consistency with no overhead (if a solution exists). Hence, it is not necessary to employ a non-polynomial algorithm (like the one proposed in [15]) for those updates. In §V, we present an algorithm that computes FU-only updates in polynomial time.



(a1) GPIA sequence, step 1 (a2) GPIA sequence, step 2

(b1) GPIA+FED sequence, step 1 (b2) GPIA+FED sequence, step 2 (b3) GPIA+FED sequence, step 3 (b4) GPIA+FED sequence, step 4

Fig. 5. Application of our procedure to the example in Fig. 2. Fig. 5(a1) and 5(a2) reports the node update sequence as computed by GPIA (phase 1 of our procedure). Fig. 5(b1), 5(b2), 5(b3) and 5(b4) depicts the final operational sequence computed after the application of FED (phase 2 of our procedure).

Previous algorithms cannot be used in the presence of an FA *control-plane:* Indeed, they do not cater for remote effects of FA control-planes (as the ones shown in Fig. 3). Moreover, §IV-B highlights the need for structurally-different approaches for FU-only and FA-involving updates. For example, for FA-involving updates, it is unsafe to rely on a static dependency graph (formalizing dependencies between update operations), calculated by exploring the pre- and post-update states, as proposed in [11]: Indeed, such a dependency graph generally changes after every update operation.

V. Provably-Safe Generic Updates

We achieve safe updates for generic networks in two phases. The first phase (§V-A) consists in running the GPIA algorithm to compute sequences of per-node FIB replacements that avoids path inconsistencies without requiring extra space in nodes' FIBs. GPIA also parallelizes replacement operations whenever safe. Since it might be impossible to complete the update using replacements alone (see Fig. 2), the second phase selectively duplicates FIB entries for the problematic destinations, relying on the FED algorithm (§V-B). This way, FED guarantees that *we always find a feasible update sequence* as long as a single FIB entry can be used on every node.

Fig. 5 illustrates the application of our two-phase procedure to the example in Fig. 2. Fig. 5(a1) and 5(a2) depict the sequence computed by GPIA, during the first phase. Despite being maximal in terms of updated nodes, the GPIA sequence does not complete the update. Figures 5(b1)–5(b4) display the final sequence computed by our two-phase procedure, where the safe node updates computed by GPIA are complemented by FIB duplication at u, v, and z.

We discuss generality and guarantees of our procedure in §V-C, providing the corresponding proofs in the appendix.

A. Phase 1: Computing Node Updates With GPIA

GPIA computes sequences of node updates iteratively. At each iteration, it simulates the update of every node not yet updated, stores nodes that do not create path inconsistencies, and updates one or more of them. It iterates until either the computed sequence is provably of maximal length, or no node can be safely updated. In the former case, we have reached the optimum, hence we return the sequence. In the latter case, GPIA applies different actions depending on the involved control-planes. Namely, for FA-involving updates, GPIA backtracks on previous choices, so that it can explore all safe update sequences. In contrast, for FU-only updates, GPIA never backtracks, degenerating into a greedy algorithm.

Input: node r, destination d, initial FIB entries fib_0 , final FIB entries fib_f, updated nodes M
Output: True if r can be updated without creating path inconsistencies

1: **check_consistency**(r, d, fib_0, fib_f, M)
2: $nhs \leftarrow compute\_next\_hops(M \cup \{r\}, fib_0, fib_f, d)$
3: $curr\_paths \leftarrow compute\_forwarding\_paths(nhs)$
4: **return** $curr\_paths = compute\_forwarding\_paths(fib_0) \vee curr\_paths = compute\_forwarding\_paths(fib_f)$

Fig. 6. Sub-procedure to check absence of path inconsistencies for d if a node r is updated.

We now provide a more detailed description of GPIA.

A basic building block of GPIA is the *check_consistency* sub-procedure that checks whether the update of a single node would cause a path inconsistency. This sub-procedure is reported in Fig. 6. Given a node r and a destination d, it computes all the forwarding paths for d that would be installed if r is updated (line 3). Then, it compares the obtained paths with both the initial and final ones (line 4). Paths are built by concatenating the next-hops used by each node. In turn, given a set of updated nodes M, the next-hops of every node are calculated by the sub-procedure *compute_next_hops* (line 2), which is polymorphic with respect to the control-planes used by each node. For any node n whose FIB entries are provided by an FU control-plane, computing its next-hops for any destination d is easy: By Property 1, if n is updated, then it uses its final next-hops for d, otherwise n uses its initial ones. In contrast, for any node m whose FIB entries are provided by an FA control-plane, we must compute its next-hops according to Property 2: For any destination d, m's next-hops are the successors of m on the shortest path from m to d such that all the nodes in the path also use the same FA control-plane as m. This computation can be done in polynomial time, by calculating all shortest paths

Input: all nodes N, initial FIB entries fib_0, final FIB entries fib_f, all destinations D, set of updated nodes M.
Output: max sequence of node updates.

1: **compute_sequence**(N, fib_0, fib_f, D, M)
2: $seq, max\_seq \leftarrow []$
3: **for** $d \in D$ **do**
4: $C_d \leftarrow \emptyset$
5: **for** $x \in N \setminus M$ **do**
6: **if** check_consistency(x, d, fib_0, fib_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$(N, fib_0, fib_f, D, M \cup \{n\})$
14: $seq \leftarrow$ add_node_update$(n, tail, C)$
15: **if** is_maximal_update(N, M, seq) **then**
16: **return** seq
17: **end if**
18: $max_seq \leftarrow$get_longer_sequence(max_seq, seq)
19: **end for**
20: **return** max_seq

Fig. 7. GPIA algorithm.

from m to d in a graph including only the nodes using the same FA control-plane as m.

The core of the GPIA algorithm is reported in Fig. 7. After variable initialization (line 2), it builds a set C_d for every destination d. This set contains all the *per-destination candidates*, i.e., all the nodes that can be updated without generating a path inconsistency for d. Sets C_d are computed by the $check\_consistency$ sub-procedure (lines 3-10). After all sets C_d are populated, the algorithm calculates *cross-destination candidates* C as the intersection of all per-destination candidate sets (line 11). Then, GPIA updates a cross-destination candidate and recurs on the remaining non-updated nodes: By iterating on all cross-destination candidates (backtracking phase), it explores all safe update sequences and eventually returns the longest one, unless it discovers a sequence that is guaranteed to be maximal (lines 12-20). To assess whether a sequence is maximal, the sub-procedure $is\_maximal\_update$ checks whether the input sequence seq covers all the nodes not yet updated at the current iteration (i.e., in $N \setminus M$).

This default GPIA behavior is actually tweaked in the case of FU-only updates. First, $is\_maximal\_update$ always returns True for FU-only updates, ensuring that GPIA never backtracks on previous candidate choices and becomes a greedy algorithm. This has no impact on the output of GPIA: Indeed, Theorem 1 implies that if a node r is a cross-destination candidate at any iteration i, it remains so until it is updated. Second, GPIA parallelizes node updates by logging cross-destination candidate sets and aggregating update operations on nodes belonging to the same cross-destination candidate set within the add_node_update sub-procedure. Theorem 1 ensures that this aggregation is safe.

As an illustration of a GPIA execution, consider again the update scenario illustrated in Fig. 2. In the first iteration, GPIA computes a cross-destination candidate set $C = \{x, y, w\}$, as they are the nodes that can be updated without triggering path inconsistencies. GPIA then updates any of the three, and iterates. Let's assume, without loss of generality, that x is updated first. In this case, the cross-destination candidate set computed by GPIA in the second iteration is $C' = \{y, w, h\}$, since the update of h would not create path inconsistencies anymore, given that x is already updated. GPIA then selects any other node in C' and iterates again. After updating x, y, w and h, GPIA finds no additional node that can be safely updated. Since $is\_maximal\_update$ always returns true for FU-only updates, GPIA terminates without exploring other update sequences (e.g., trying to update y first). Moreover, add_node_update aggregates node updates into groups, according to the sequence of explored candidate sets. In particular, it aggregates the updates of x, y and w in a single update step, since the first candidate set computed by GPIA is $\{x, y, w\}$. Eventually, GPIA returns $[\{x, y, w\}, \{h\}]$ as update sequence, as shown in Figs. 5(a1) and 5(a2).

Note that we deliberately omitted several algorithmic optimizations, including those added to our GPIA implementation. For instance, forwarding paths can be stored, incrementally recomputed and re-used across iterations. A detailed description of these optimizations is beyond the scope of this paper.

Input: all nodes N, initial FIB entries fib_0, final FIB entries fib_f, all destinations D, updated nodes M, node-update sequence S.
Output: final operational sequence.

1: **compute_FIB_duplication**$(N, fib_0, fib_f, D, M, S)$
2: $U \leftarrow \emptyset$
3: $S' \leftarrow S$
4: $N' \leftarrow N \setminus M$
5: **for** $(x, d) \in N' \times D$ **do**
6: **if** check_consistency(x, d, fib_0, fib_f, M) = False and is_guaranteed_final(x, d, fib_0, fib_f, M) = False **then**
7: $U \leftarrow U \cup \{(x, d)\}$
8: **end if**
9: **end for**
10: $B \leftarrow$ get_per_dest_border_nodes (fib_0, fib_f, U)
11: $S' \leftarrow$ add_prepare_match_tag(S', B, N')
12: $O_1 \leftarrow$ swap_tags(B)
13: $O_2 \leftarrow$ update_nodes(N')
14: $O_3 \leftarrow$ clean_border_nodes(B)
15: $S' \leftarrow$ get_concat(S', O_1, O_2, O_3)
16: **return** S'

Fig. 8. FED algorithm.

B. Phase 2: Selectively Duplicating FIB Entries With FED

The second phase of our update-computation procedure consists in selectively duplicating the FIB entries (of some nodes, for specific destinations) that cannot be replaced without creating path inconsistencies.

The algorithm used in this second phase is called FED, and is detailed in Fig. 8. It is basically a generalization of previous techniques based on maintaining both the initial and final FIB entries on all nodes at the same time [2], [17].

After initialization (lines 2-3), FED identifies all the *unsafe pairs*, that is, node-destination pairs for which the FIB entries

have to be duplicated. Given a sequence S produced by GPIA, FED first computes the set N' of *non-ordered nodes*, i.e., the nodes that are not included in the GPIA sequence (line 4). It then constructs the set U of unsafe pairs by iterating on all non-ordered nodes and destinations (lines 5-9). For each non-ordered node n and destination d, FED adds (n, d) to U, unless (i) n can be safely updated for d, i.e., $consistency\_check$ returns true; or (ii) n is guaranteed to have the same next-hops in the initial and final state, i.e., $is\_guaranteed\_final$ returns true, because n's initial and final FIB entries are identical and no FA control-plane is involved in the update.

Based on U, FED duplicates FIB entries *only for the identified unsafe pairs* adding some steps to the input sequence S and returning a modified sequence S' (lines 10-16).

First, $add\_prepare\_match\_tag$ adds one set of parallel operations to the first step in S, to prepare packet tagging and tag matching. Namely, for each unsafe pair (n, d), the initial FIB entry of n for d is replaced with two entries. The first replacing entry matches a tag d_init (e.g., no tag or a tag already present in current packets) and applies the initial forwarding action (e.g., forwarding packets to the initial next-hops of n). Similarly, the second replacing entry matches a $d\_fin \neq d\_init$ tag and applies the final action (e.g., forwarding packets to the final next-hops). Tags for any unsafe destination d are set by border nodes, i.e., the ingress nodes of packets for d – computed by the $get\_per\_dest\_border\_nodes$ sub-procedure (line 10). In the first update step of S', border nodes are instructed to push d_init tags. In the example of Fig. 5, for instance, FED imposes that additional operations are performed in parallel to the update of x, y, and w (as in the first step of the GPIA sequence). Those additional operations (i) duplicate the FIB entries of u, v, and z for d to match the d_init and d_fin tags in addition to the destination; and (ii) instruct u and h (from which packets for d enter the network) to add the d_init tag to packets for d. Globally, all those operations lead to the state displayed in Fig. 5(b1).

In addition, FED adds three successive sets O_1, O_2, O_3 of parallel operations at the very end of S (lines 14-17 in Fig. 8). Operations in O_1 instruct border nodes to push final d_fin tags. In Fig. 5, this translates into u pushing the d_fin tag rather than the d_init one to packets for d after the GPIA sequence has been applied. Note that u also starts sending those packets to its final next-hop z, as a result of matching tags for d since the first update step. The resulting state is reported in Fig. 5(b3). Operations in O_2 install final FIB entries on all non-ordered nodes. Notably, the entries installed for any unsafe node-destination pair (n, d) are replaced by the final entry of n for d. In our example in Fig. 5, the application of operations in O_2 brings v and z to their final state (see Fig. 5(b4)), and removes FIB entry duplication on u (but u keeps tagging packets for d). The update is virtually over after O_2. Operations in O_3 remove packet tagging.

C. Properties of Our Two-Phase Procedure

(i) Our procedure applies to any combination of coexisting control-planes: Some GPIA sub-procedures are polymorphic with respect to the control-planes involved in the update, as discussed in §V-A. This way, the resulting sequence is safely applicable to networks running arbitrary control-planes. FED is also general. Indeed, the critical operations that FED adds to GPIA sequences, i.e., packet tagging and tag matching, are supported by both SDN and IGP nodes. For SDN, the definition itself of nodes as programmable hardware (e.g., through low-level protocols like OpenFlow) natively provides support for those operations [2]. In contrast, IGP nodes can rely on mechanisms like Policy Based Routing (PBR), commonly available on commercial devices (see, e.g., [25], [26]).

(ii) Our procedure computes safe updates: Indeed, we proved the following theorem.

Theorem 3: Our two-phase procedure ensures that path inconsistencies do not occur during the update.

(iii) Our procedure updates the maximum number of nodes without adding FIB entries: Indeed, we proved the following theorem.

Theorem 4: GPIA finds update sequences in which the maximal number of nodes are updated without causing path inconsistencies.

As a consequence of Theorem 1, additional properties hold for **FU-only updates**.

(iv) The time complexity of our procedure is polynomial: In general, the time efficiency of GPIA depends on the backtracking phase, which can lead to exponential complexity. However, for FU-only updates GPIA never backtracks (see §V-A). Theorem 1 ensures that this has no impact on the safety and length of the computed update sequence.

(v) Our procedure computes quick updates: Indeed, for FU-only updates, GPIA minimizes the length of the computed sequence by parallelizing operations (see Theorem 2). Moreover, FED adds three groups of operations to the sequence produced by GPIA (if it does not complete the update). Our evaluation (§VI) experimentally confirms that the computed sequences tend to have a very limited number of steps.

VI. Evaluation

In this section, we experimentally evaluate the effectiveness of our two-phase procedure (§V), focusing on the gain that it achieves with respect to previous approaches in practical use cases. We describe our evaluation setup in §VI-A. We report experimental results and compare our approach with mainstream alternatives in §VI-B and §VI-C.

For the evaluation, we used a single-threaded Python implementation of the algorithms reported in Figs. 6, 7 and 8. We have publicly released this implementation at *http://inl.info.ucl.ac.be/softwares/hybridupdate*.

A. Dataset and Methodology

We used realistic topologies: Our dataset included all the publicly available Rocketfuel topologies [27].

In our experiments, we used border nodes as destinations of the forwarding paths to be updated. Those nodes attract Internet traffic and are usually critical for both traffic engineering and policy routing requirements. We determined the border nodes as follows. For each topology, we partitioned the nodes according to the cities in which they are located. Then, we defined the nodes having no direct links with nodes

in other cities as border nodes. The identified border nodes are 38 out of 306 nodes for AS 1221, 212/315 for AS 1239, 10/322 for AS 1755, 27/656 for AS 3257, 19/294 for AS 3967, and 73/748 for AS 6461.

We relied on realistic update scenarios: We performed multiple experiments covering two scenarios.

The first scenario consisted in *single-destination* updates where we changed forwarding paths for a single destination. This scenario mimics fine-grained update operations like per-destination deployment of new control-planes (e.g., SDN), virtual machine migrations from one data center to another, and update of low-delay flows (e.g., towards a VoIP server). In each of those experiments, we considered a Rocketfuel topology and we randomly selected one border node as destination.

The second scenario reflected *multi-destination* updates in which all the border nodes are simultaneously selected as destinations. It intuitively maps to a worst-case analysis of large traffic engineering operations including global optimization of forwarding paths, or prompt reaction to network events (e.g., congestion-threatening traffic surges).

We computed the initial and final forwarding paths as follows. We took the shortest paths in the original topology as initial paths, installed before the update. Then, to evaluate the impact of the number of updated paths, we performed two sets of experiments per scenario, in which the final paths were computed as the shortest paths in a topology with 5 or 10 modified link weights respectively. The modified link weights were assigned values consistent with the other weights in the same topology. We chose to modify 5 or 10 weights because traffic engineering optimizations typically correspond to a number of weight changes in that range [28].

We stress that our approach supports additional scenarios (not evaluated for brevity), like moving traffic flows away from specific links and nodes to avoid or remove congestion.

We considered our two-step procedure as well as alternative approaches: Consistently with Table I, we grouped alternatives to our approach into two broad classes.

The first class is *two-phase commit*, e.g., used in [2], [5], [6], and [17]. This approach duplicates all FIB entries so that each node maintains both the initial and final FIBs. The update then consists in tagging packets so that the final FIB rules are applied network-wide, and finally removing the initial FIB entries. Since FED is a generalization of previous two-phase commit algorithms, we ran FED on an empty sequence to compare our procedure with two-phase commit.

The second class is *FIB-entry replacement ordering*, where FIB entries are replaced in a carefully-computed order. To compute a safe order, several algorithms (e.g., [8], [15]) have been proposed in literature. To compare with such algorithms, we ran GPIA alone: Since GPIA is provably safe and optimal (see Theorem 4), no algorithm based on FIB-entry replacement ordering can perform better than GPIA.

We performed simulations: The compared algorithms are correct and applicable to real networks. Indeed, safety of the computed sequences is proved in the appendix, while applicability is ensured by the ability of our control-plane model to capture the behavior of real control-planes and devices, as shown in [13] with emulations and testbeds.

The goal of our evaluation is therefore to assess the practicality of our two-step procedure in comparison to existing techniques. Consistently, we use success rate, memory overhead, time to compute the update sequence, and length of computed sequences as evaluation metrics.

With these observations in mind, we opted for simulating the application of the evaluated techniques to our update scenarios. With respect to emulations, simulations are (i) more general, since we can generalize results to any FU or FA control-plane irrespectively of low-level environmental (e.g., event timings) and implementation (e.g., protocol message format and device-specific operations) details [13]; (ii) more scalable, enabling us to evaluate different update techniques on realistic networks with hundreds of nodes; (iii) independent from the experimental setup, from the virtualization environment to the software emulating the device operating system; and, consequently, (iv) easier to reproduce.

To ensure statistical significance, we repeated every experiment 60 times. In each repetition, we randomly selected the links having different weights in the final graph.

B. Updates of FIB-Unaware Control-Planes

We started our evaluation restricting to networks exclusively running FU control-planes both before and after the simulated update. We recall that FU control-planes prominently include commonly-used link-state IGPs and OpenFlow controllers. Hence, the evaluated update scenarios support all our use cases (see §II) in hybrid networks running a link-state IGP and an FU SDN controller. Moreover, they map to arbitrary IGP configuration modifications, and change of FIB rules in pure SDN networks.

Experimental results show that our procedure always achieves safe updates with few additional FIB entries.

Contrary to FIB-entry replacement ordering, our procedure achieves 100% success rate: Fig. 9 compares the percentage of cases in which our approach and any entry replacement ordering technique can compute a safe update.

Fig. 9(a) shows that techniques only using FIB-entry replacements fail to compute a safe update in many cases (from 25% to 90% depending on the topology) in single-destination experiments. Those techniques quickly become practically inapplicable if multiple destinations are impacted by the update (see Figs. 9(b) and 9(c)): Their success rate drops under 20% and 5% for all topologies in our multi-destination experiments with 5 or 10 different weights respectively.

Globally, Fig. 9 suggests that solely relying on replacement ordering is not practical if strong consistency has to be guaranteed. In contrast, our procedure (as well as two-phase commit) is successful in any update scenario. This is because our approach complements FIB-entry replacements with FIB-entry duplications, i.e., running FED after GPIA.

Our procedure hugely reduces the number of duplicated FIB entries: A FIB-entry replacement ordering is seldom sufficient to perform a complete update. However, in many cases, we can update a significant number of nodes by just

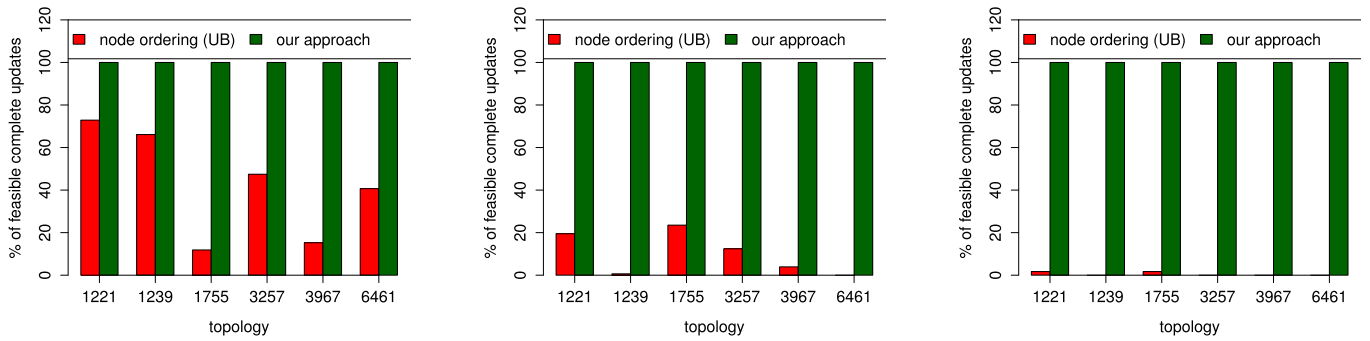


Fig. 9. Contrary to FIB replacement techniques, our approach can always be used for a safe update. (a) Single-destination. (b) Multi-destination, 5 link changes. (c) Multi-destination, 10 link changes.

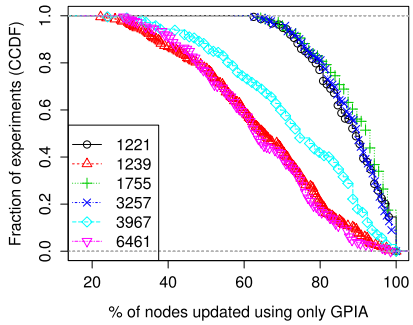


Fig. 10. Many nodes can be safely updated without duplicating any FIB entry even in the multi-destination scenario with 10 link changes.

ordering FIB-entry replacements. Fig. 10 shows the distribution (as complementary CDF) of the percentage of nodes that can be updated without applying any FIB entry duplication. A point (x, y) in the figure means that at least $x\%$ nodes can be safely updated without FIB duplication in $y * 100\%$ of our experiments. As the plot shows, for 3 topologies out of 6 GPIA can update more than 80% of the nodes without FIB duplication, in more than 80% of the experiments. In the remaining topologies, a lower but still significant percentage of nodes (more than 60% in 60% of the experiments) can also be safely updated without duplication.

Our procedure leverages this opportunity to significantly reduce the amount of resources consumed during the update, i.e., updating a signicant fraction of the network (see Fig. 10) by just replacing FIB entries, with no overhead.

Fig. 11 shows that our procedure hugely reduces the number of additional FIB entries across all scenarios and topologies. It indeed displays the additional FIB entries that are required in our experiments by both our procedure and two-phase commit techniques. The figure includes three plots, for the single- and the multi-destination scenarios respectively. In each plot, an x value corresponds to a single experiment. Experiments are grouped on the x axis by the used topology. The y axis reports the additional FIB entries required for the x-th experiment, in percentage with respect to the initial number of FIB entries. The plots show that our procedure duplicates a percentage of the initial FIB entries close to zero in many cases and less than 20% in all the experiments, with a very limited variance across topologies and scenarios (as highlighted by the tendency of circles to be close to each other). In other words, it achieves a reduction of the duplicated FIB entries between 80% and 100% with respect to two-phase commit techniques.

In addition, our procedure requires no FIB duplication at all on many nodes. Fig. 12 shows the distribution of the FIB entries duplicated by our procedure on every node in the different experiments. The plot refers to the multi-destination scenario with 10 link weight changes. Less than 1% of the initial FIB is duplicated on 50% of the nodes on all the topologies. Moreover, 75% of the nodes (upper border of the boxes) require less than 5% additional FIB entries in all the cases except 3967. Overall, only in a very few cases (5% of the nodes across all the experiments), more than 50% FIB entries are duplicated for 1755 and 3967, and more than 20% for all the other topologies. That is, heavy FIB entry duplication is very localized to few nodes which likely produce intermediate paths for most destinations. This localization decreases the likelihood that stringent hardware requirements of legacy nodes traversed by few paths can hamper the update feasibility, contrary to two-phase commit techniques (see example in §III).

Our approach quickly computes short update sequences: We used the number of update steps, i.e., the length of the produced sequences, as an abstract measure of the time required to complete a safe update. Moreover, to measure how long our procedure takes to compute an update sequence, we tracked the computation time of our single-threaded Python prototype. A summary of both the update and the computation times collected during our multi-destination experiments (with 10 modified link weights) is reported in Table II.

The update sequences computed by our approach consisted in few steps of parallel operations, i.e., up to 6 steps for 90% of the experiments and at most 9 across all of them. This is a small increase with respect to a standard two-phase commit approach, which always requires the 4 steps illustrated in Fig. 8. As a comparison, consider the alternative solution, proposed in [5], to reduce the number of duplicated FIB entries by splitting the update in multiple rounds, each updating a subset of the affected destinations. This solution increases the number of update steps by a multiplication factor that is proportional to the number of rounds. To achieve reductions

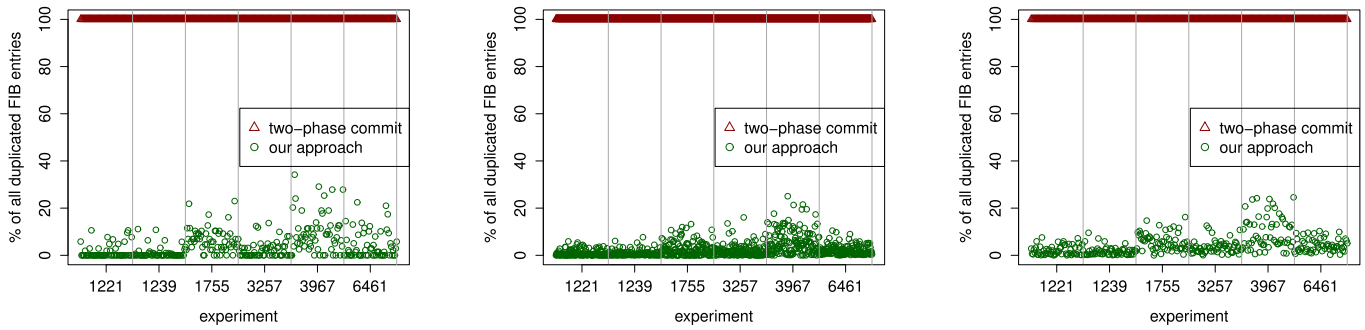


Fig. 11. Contrary to two-phase commit techniques, our approach uses a limited amount of network resources (additional FIB entries). (a) Single-destination. (b) Multi-destination, 5 link changes. (c) Multi-destination, 10 link changes.

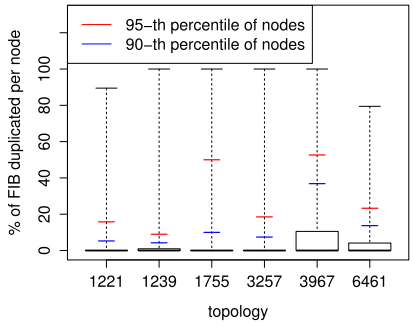


Fig. 12. For the vast majority of the nodes, our approach duplicates a number of FIB entries equal or close to zero.

TABLE II

OUR SIMULATIONS SHOW THAT OUR PROCEDURE COMPUTES FAST UPDATES IN A SHORT TIME

Topology	*Update Steps*	*Exe time*
1239 (Median)	5	15.82 sec
1239 (90-th perc)	5	21.25 sec
1239 (Max)	6	26.97 sec
Others (Median)	5	1.61 sec
Others (90-th perc)	6	4.62 sec
Others (Max)	9	7.33 sec

of additional FIB entries comparable to our approach, such a factor can range between 8 and 15 as shown by [5, Fig. 9], resulting in update sequences with 30 to 60 steps.

Computation time is also quite low. It is few seconds in the worst case for all the topologies but 1239 which is the biggest topology for both number of nodes (more than 300), links and destinations (more than 200). Even on this topology, however, the computation time is still lower than 30 seconds in all our experiments. While those computation times are already good to support a wide range use cases (including most of those presented in §II), a faster update computation may be required in specific cases (e.g., to quickly react to a link failure or a security threat). To this end, both the algorithms shown in Figs. 7 and 8 and their implementation can be largely optimized, e.g., avoiding to recompute all the forwarding paths for each consistency check and parallelizing the computation of the cross-destination candidate set C.

C. Updates With FIB-Aware Control-Planes

We now evaluate how our algorithms perform in scenarios not covered by prior works. We indeed repeated the multi-destination experiments with 10 link weight changes assuming that the initial and final states are provided by an FA and an FU control-planes respectively. Such scenarios correspond to use cases including incremental deployment of SDN in an enterprise network running EIGRP [24] and the replacement of a FA SDN controller with a FU one.

For the sake of realism, we cut the execution of our procedure after a maximum amount of backtracking that we call *cut value*. Indeed, for FA-involving scenarios, the GPIA algorithm implementing the first step of our procedure adopts a brute-force strategy, i.e., backtracking on previous choices to generate all possible safe per-node update sequences. This would result in computations that are too long for many update use cases. For instance, even if only 30 nodes (i.e., less than 10% of the 315 nodes in the 1239 topology) can be safely updated in any relative order, GPIA will generate all the permutations of those 30 nodes. For this reason, we limited GPIA to never backtrack a number of times greater than the cut value. By definition, higher cut values correspond to lower update overhead but also higher computation time. To experimentally evaluate trends, we repeated each experiments with cut values ranging from 1 to 3. We stopped at 3 since results rarely differ between values 2 and 3 in our experiments.

We can only compare with two-phase commit techniques: Indeed, GPIA is the first FIB-entry replacement algorithm that guarantees safe updates in the presence of an FA control-plane. Moreover, as already noted in §IV-C, extending previous FIB-entry replacement algorithms to FA-involving updates is hard.

Our procedure significantly reduces duplicated FIB entries, whenever safe: Fig. 13 displays a quantitative comparison between our procedure and the generalized two-phase commit one. In particular, the figure plots the total percentage of FIB entries that are duplicated by our approach (circles, vertical crosses, or oblique crosses depending on the specific cut value used in the experiment) and the two-phase commit one (triangles) for every experiment. The overhead reduction, however, heavily depends on the network topology in this scenario: Intuitively, larger and more-connected networks are harder to

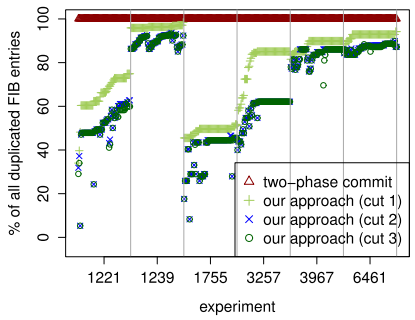


Fig. 13. The generality of our approach enables support for safe updates including FA control-planes like distance-vector IGPs and FIB-reacting SDN controllers.

update because their higher number of paths makes remote effects of node updates more likely. Indeed, the overhead remains quite large for updates on 1239, but it is much more limited for smaller topologies like 1221 and 1755. In the latter cases, our procedure reduces the number of duplicated FIB entries by about $60-80\%$ in many experiments, with a maximum of more than 90% for both topologies. Note that the variance of our results for a single topology is more limited (as shown by the tendency of points to aggregate in clusters).

Our experiments quantify intrinsic complexity of FA*-involving updates:* In general, Fig. 13 shows that the overhead reduction achieved by our procedure is more limited than for FU-only updates (see Fig. 11(c) for comparison), despite increasing cut values. Those results suggest that the limited performance of our procedure depends more on the presence of the FA control-plane, rather than (say) on the cut value. In some update scenarios, the remote effects of single-node updates induced by the FA control-plane are so constraining, that only few FIB entry replacements can be ordered and FIB entries must be duplicated on most of the nodes (see results for 1239 in Fig. 13). In comparison with FU-only updates, both (i) the ordering phase of our approach (i.e., GPIA) is harder to compute (see §III), and (ii) the FIB duplication phase (i.e., FED) generates significantly-higher overhead.

VII. Conclusions

In this paper, we studied the problem of computing practical operational sequences for network updates, so that strong consistency is guaranteed, with low overhead and limited update time. We also generalized previous works by making no assumption on the type and number of the control-planes involved in the update. This also allowed us to cover compelling use cases, like incremental SDN deployment and management of hybrid SDN networks, not supported by previous works.

We related the computational complexity of finding a safe update sequence to intrinsic properties of the involved control planes. Moreover, we analyzed the inherent limitations of the approaches proposed by previous works, and revisited them in the light of our theoretical contributions. Based on the gained insight, we proposed a procedure that combines the strengths of previous techniques. We proved that our procedure computes safe operational sequences that (i) update the maximal number of nodes that can be updated without overhead, in the minimum number of steps; and (ii) surgically add overhead when necessary to update the remaining nodes. A thorough evaluation based on realistic data and scenarios confirms that our procedure outperforms mainstream techniques. Indeed, our algorithms always find a safe operational sequence even when FIB-entry replacement techniques cannot, and decrease the update overhead by $80-100\%$ with respect to two-phase commit approaches.

Appendix

Notation: In the proofs, we rely on the following notation. We denote the next-hop of a node x for a destination d at time t as $next(x,d,t)$, and the forwarding path from x to d at t as $\pi(x,d,t)$. A forwarding path is defined as a concatenation of next-hops, that is, $\pi(x,d,t) = (v_0 \dots v_k)$, with $k \geq 0$, $v_0 = x$, $v_k = d$, and $\forall i = 1,\dots,k$ $v_i = next(v_{i-1},d,t)$. Note that those definitions admit that nodes forwards packets of the same (TCP) flow to multiple next-hops and over distinct paths. We indeed define destinations as sub-flow identifiers such that all nodes forward packets with that identifier over a single path. Finally, we consider discrete time. Each time instant corresponds to a step in our update procedure. Special values 0 and f indicate the initial and final times, that is, before starting and after completing the update respectively.

A naive strategy is correct and optimal for FU*-only updates:* We indeed prove Theorems 1 and 2.

Theorem 1: In an FU-only update, if a node r can be safely updated at time t, r can also be safely updated at any $t' > t$.

Proof: Assume by contradiction that there exists a node r and a time t so that r can be safely updated at a given time t, but it cannot at time $t+1$.

Let S be the set of nodes updated at step $t+1$. By hypothesis, at $t+1$, updating r would trigger a path inconsistency for at least one destination d. Since the update is FU-only, Property 1 holds. Consequently, for r not to be safely updatable at $t+1$, the final next-hop x of r is such that $x \in S$ and $next(x,d,0) \neq next(x,d,f)$. Also by Property 1, updating r only changes the next-hops of r, so we must have $next(r,d,0) \neq next(r,d,f)$. Then, one of the following cases must hold.

- $r \in \pi(x,d,t+1)$. Then, consider time $t+1$. Since x is updated at $t+1$, $next(x,d,t+1) \neq next(x,d,0)$, which implies $\pi(x,d,t+1) \neq \pi(x,d,0)$. Moreover, r is not updated yet, hence $next(r,d,t+1) \neq next(r,d,f)$, which implies $\pi(x,d,t+1) \neq \pi(x,d,f)$. That is, the path of x for d at $t+1$ is different from both its initial and final paths for d, i.e., a path inconsistency at x.
- $x \in \pi(r,d,t+1)$. Consider again time $t+1$. Since x is updated at $t+1$, $next(x,d,t+1) \neq next(x,d,0)$, which implies $\pi(r,d,t+1) \neq \pi(r,d,0)$. Moreover, since r is not updated at $t+1$, $next(r,d,t+1) \neq next(r,d,f)$, implying $\pi(r,d,t+1) \neq \pi(r,d,f)$. In other words, the path of r for d at $t+1$ is different from both its initial and final paths for d, which is a path inconsistency at r.

In both cases, updating $x \in S$ at time t triggers a path inconsistency at $t+1$, hence contradicting the hypothesis that x is safely updated at t. ■

Theorem 2: For FU-only updates, the greedy strategy computes safe sequences of minimal length.

Proof: Consider any FU-only update and let S be the sequence computed by the greedy strategy described in §IV-B. Assume by contradiction that there exists a safe sequence S^* which has less steps than S. This implies that at least one node x is updated at a step j in S and a step $i < j$ in S^*. We have two cases.

- x can be safely updated after the node updates performed in S until i. By definition, the greedy strategy would have updated x at i, in this case, hence contradicting the definition of x.
- x cannot be safely updated after the node updates performed in S until i. Since S^* is safe and x is updated at i in S^*, there must exist a node y that is updated in S^* at step $z < i$, and in S at $k \geq i$.

In the first case, we directly generate a contradiction. In the second case, we can iterate the same argument applied to x on y. During this iteration, we consider a node (i.e., y) that is updated in S^* before the last one considered (i.e., x). Since S^* is finite, we eventually end up in the first case, that yields the statement. ∎

Our two-step procedure is correct and efficient: We now prove Theorems 3 and 4.

Theorem 3: Our two-step procedure ensures that path inconsistencies do not occur during the update.

Proof: Consider any sequence S generated by our procedure. We now show that any packet for any destination d provably follows either the initial or the final path from the border node b to d.

For any time before b uses the final tag, path consistency is directly guaranteed by GPIA (see Theorem 4). Indeed, non-ordered nodes forwards always to their initial next-hops as assumed in GPIA.

Consider now any time t during the update in which b uses the final tags. For each node $n \in \pi(b, d, t)$, we have two cases. If n is a node appearing in the GPIA sequence, then n must be already updated. Otherwise, by definition of the duplication algorithm (see Fig. 8), n either (i) has the same next-hop before and after the update (as ensured by the $is\_guaranteed\_final$ function in Fig. 8), (ii) is updated, or (iii) matches the final tag. In all cases, n uses its final next-hop for d. Hence, $\pi(b, d, t) = \pi(b, d, f)$, which yields the statement. ∎

Theorem 4: GPIA finds update sequences in which the maximal number of nodes are updated without causing path inconsistencies.

Proof: 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 sequences is ensured by backtracking. ∎

Generality of our approach for hybridly-controlled flows: For simplicity, our two-step procedure is defined in the absence of hybridly-controlled flows (see §IV-A). We now illustrate a *zero-overhead pre-update transformation*, to be applied before our update computation procedure to hybridly-controlled flows. It changes the path of every hybridly-controlled flow into an *equivalent* one entirely determined by an SDN controller. To this end, the SDN controller overwrites IGP FIB entries of non-SDN nodes, by installing routes pointing to the same next-hops as the replaced FIB entries. Such overwriting is performed following a topological-sort order, so that no FIB entry is overwritten on a node still used by nodes not processed yet. The following theorem proves that this is safe, as pre- and post-transformation next-hops remain the same for all nodes.

Theorem 5: The pre-update transformation guarantees strong consistence.

Proof: Assume by contradiction that an intermediate path $P = (s \dots d)$ is triggered by a pre-update transformation. Since all pre- and post-transformation paths are the same, P cannot be due to a combination of initial and final next-hops. Hence, for P to be an intermediate path, there must exist a node $x \in P$ that uses a next-hop that it is not using initially nor finally. Property 1 ensures that it is not possible for FU-only networks. If a FA control-plane is involved, then P can be raised only if the initial next-hops of x for d have temporarily withdrawn their FA route for d. In turn, this is only possible if the FIB entries of some node between x and d are overwritten before x's ones. However, the transformation never overwrites a node used by another, by definition, hence yielding a contradiction. ∎

Moreover, the pre-update transformation is *always possible*, since the controller can always install the post-transformation routes by relying on an additional link-state IGP with a tweaked augmented topology [22].

REFERENCES

[1] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. SIGCOMM*, 2013, pp. 3–14.

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

[3] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Proc. HotSDN*, 2012, pp. 67–72.

[4] K.-T. Foerster, S. Schmid, and S. Vissicchio. "Survey of consistent network updates." accessed on Sep. 2016. [Online]. Available: https://arxiv.org/abs/1609.02305

[5] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. HotSDN*, 2013, pp. 49–54.

[6] X. Jin *et al.*, "Dynamic scheduling of network updates," in *Proc. SIGCOMM*, 2014, pp. 539–550.

[7] R. McGeer, "A safe, efficient update protocol for openflow networks," in *Proc. HotSDN*, 2012, pp. 61–66.

[8] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Seamless network-wide IGP migrations," in *Proc. SIGCOMM*, 2011, pp. 314–325.

[9] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. SIGCOMM*, 2013, pp. 15–26.

[10] H. Liu *et al.*, "zUpdate: Updating data center networks with zero loss," in *Proc. SIGCOMM*, 2013, pp. 411–422.

[11] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proc. HotNets*, 2013, Art. no. 20.

[12] K.-T. Förster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *Proc. IFIP Netw.*, 2016, pp. 1–9.

[13] S. Vissicchio, L. Cittadini, O. Bonaventure, G. G. Xie, and L. Vanbever, "On the co-existence of distributed and centralized routing control-planes," in *Proc. IEEE INFOCOM*, 2015, pp. 469–477.

[14] A. Ludwig, M. Rost, D. Foucard, and S. Schmid, "Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies," in *Proc. HotNets*, 2014, pp. 15:1–15:7.

[15] J. McClurg, H. Hojjat, P. Černý, and N. Foster, "Efficient synthesis of network updates," in *Proc. PLDI*, 2015, pp. 196–207.

[16] P. Francois and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *IEEE/ACM Trans. Netw.*, vol. 15, no. 6, pp. 1280–1292, Dec. 2007.
[17] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," in *Proc. SIGCOMM*, 2008, pp. 111–122.
[18] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, Apr. 2014.
[19] S. Vissicchio and L. Cittadini, "FLIP the (Flow) table: Fast lightweight policy-preserving SDN updates," in *Proc. INFOCOM*, Apr. 2016, pp. 1–9.
[20] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. INFOCOM*, Apr. 2013, pp. 2211–2219.
[21] A. Atlas *et al.*, *An Architecture for the Interface to the Routing System*, document RFC 7921, IETF, Jun. 2016.
[22] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. SIGCOMM*, 2015, pp. 43–56.
[23] Z. A. Qazi *et al.*, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *Proc. SIGCOMM*, 2013, pp. 27–38.
[24] D. A. Maltz *et al.*, "Routing design in operational networks: A look from the inside," in *Proc. SIGCOMM*, 2004, pp. 27–40.
[25] Cisco Systems. *Configuring Policy-Based Routing*. accessed on Jan. 10, 2017 [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfpbr.html
[26] Juniper. *Configuring Filter-Based Forwarding to A Specific Outgoing Interface or Destination IP Address*. accessed on Jan. 10, 2017 [Online]. Available: http://www.juniper.net/techpubs/en_US/junos12.2/topics/topic-map/filter-based-forwarding-policy-based-routing.html
[27] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. SIGCOMM*, 2002, pp. 133–145.
[28] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. INFOCOM*, 2000, pp. 519–528.

Stefano Vissicchio received the master's degree and the Ph.D. degree in computer science from the Roma Tre University in 2008 and 2012, respectively. He was a Post-Doctoral Researcher with the Université catholique of Louvain. He is currently a Lecturer with University College London. His research interests span network management, routing theory and protocols, measurements, and new network architectures like software defined networking. He has received several awards, including the ACM SIGCOMM 2015 best paper award, the ICNP 2013 best paper award, and two IETF/IRTF Applied Networking Research Prizes.

Laurent Vanbever received the Ph.D. degree in computer science from the University of Louvain, Belgium, in 2012. He was a Post-Doctoral Research Associate with Princeton University, where he collaborated with Prof. J. Rexford. He has been an Assistant Professor with ETH Zurich, where he leads the Networked Systems Group, since 2015. His research interests lie at the crossroads between theory and practice, with a focus on making large network infrastructures more manageable, scalable, and secure. He has won several awards for his research, including: the USENIX NSDI 2016 Community Award, the ACM SIGCOMM 2015 best paper award, the ACM SIGCOMM Doctoral Dissertation Award (runner-up), the University of Louvain Best PhD Award, the ICNP 2013 best paper award, and three IETF/IRTF Applied Networking Research Prizes.

Luca Cittadini received the master's degree and the Ph.D. degree in computer science and automation from the Roma Tre University in 2006 and 2010, respectively, defending the thesis Understanding and Detecting BGP Instabilities. He was a teaching Assistant with the Computer Network Research Laboratory. His research activity is primarily focused on routing, and he has worked on both intra-domain and inter-domain routing topics, including theoretical analysis of the BGP protocol, configuration and reconfiguration techniques, and also active and passive measurements.

Geoffrey G. Xie received the B.S. degree in computer science from Fudan University, China, and the Ph.D. degree in computer sciences from the University of Texas at Austin. He was a Visiting Scholar with the School of Computer Science, Carnegie Mellon University, from 2003 to 2004, and the Computer Laboratory, University of Cambridge, U.K., from 2010 to 2011. He is currently a Professor and an Associated Chair of the Computer Science Department, Naval Postgraduate School. He has authored over 70 articles in various areas of networking. His current research interests include formal network analysis, routing design and theories, cloud security, and abstraction driven design of enterprise networks. He co-chaired the ACM SIGCOMM Internet Network Management Workshop in 2007. He received the best paper award at the 2007 IEEE ICNP conference and the 2011 IEEE Fred W. Ellersick Award.

Olivier Bonaventure received the degree from the University of Liège in 1992, and the Ph.D. degree in 1999, and spent one year at Alcatel in Antwerp, Belgium. He is currently a Full Professor with the Universite Catholique de Louvain, Belgium, where he leads the IP Networking Laboratory. He has authored over 80 papers and was granted several patents. He is a member of the CoNEXT Steering Committee. He received several awards, including the IEEE INFOCOM 2007 best paper award and the 2012 USENIX NSDI Community award. He serves on the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING. He currently serves as Education Director within ACM SIGCOMM.