IGP-as-a-Backup for Robust SDN Networks

Olivier Tilmans Stefano Vissicchio
Université catholique de Louvain
olivier.tilmans@student.uclouvain.be stefano.vissicchio@uclouvain.be

***Abstract*—One of the main concerns on SDN is relative to its ability to quickly react to network failures, while limiting both the control-plane overhead and the additional forwarding state kept by data-plane devices. Despite its practical importance, this concern is often overlooked in OpenFlow-based proposals.**

In this paper, we propose a new architecture, called IBSDN, in which a distributed routing protocol flanks OpenFlow to improve network robustness, reaction to failures, and controller scalability. In deeply exploring this idea, we complement our architecture with data-plane triggered mechanisms that improve its efficiency. We prove that the resulting solution ensures robustness for any combination of topological failures, and quickly reduces the path stretch. Finally, experimenting with a prototype implementation, we show that our approach is practical and overcomes the main limitations of previous work.

I. Introduction

For years, distributed protocols have been typically used to manage and operate computer networks. Contrary to this practice, the emerging Software-Defined Networking (SDN) paradigm proposes network logic centralization. In particular, SDN is based on a clear separation between the control-plane, to be moved to a logically-centralized controller, and the data-plane, to be kept on network devices (e.g., routers). By exploiting this separation, SDN promises to ease the deployment of multiple capabilities such as network virtualization [1], flexible access control [2] and fine-grained traffic engineering [3]. These capabilities, along with the growing support for SDN protocols like OpenFlow (OF) [4], have recently attracted the interest of many operators. Even device vendors are proposing SDN protocols [5] and promising their support in a short time.

Despite such an interest, SDN deployments are still limited in number, likely because of a variety of technical, economical, and organizational reasons [6]. Among the former ones, a major concerns is about robustness and failure recovery. Indeed, proposed OpenFlow-based solutions to improve robustness of SDN networks suffer intrinsic limitations that can make them unpractical, especially in geographically-distributed networks like those of large enterprises or Internet providers. Namely, previous proposals either need to add a huge number of backup entries to devices' Forwarding Information Basis (FIBs), or to fully rely on the centralized controller for failure recovery. The former proactive approach (e.g., [7]) requires much more powerful and expensive hardware than currently available OpenFlow switches [8], and does not arbitrarily scale (e.g., with respect to the number of simultaneous failures). The latter reactive approach (e.g., [9]) poses severe efficiency and scalability challenges to the controller implementation, demands for a very fast and reliable connection between devices and controller, and does not cover control-plane failures.

S. Vissicchio is a postdoctoral researcher of the Belgian fund for scientific research (F.R.S.-FNRS)

Abstractly, the main limitation of prior proposals derives from the fact that only static forwarding rules can be specified in OpenFlow. Since massive forwarding changes may be needed in the case of failures, relying on OpenFlow leads to difficult trade-offs between robustness, performance, and scalability. In contrast, distributed routing protocols enrich devices with the ability to autonomously and quickly [10] synchronize among them and locally re-route traffic. This enables effective and scalable solutions for (fast) failure recovery.

Following this high-level analysis, this paper proposes and evaluates a hybrid SDN architecture that employs a logically-centralized controller for long-term optimization and relies on distributed protocols for short-term reaction to failures. In particular, the controller uses OpenFlow to configure arbitrary forwarding paths (e.g., overcoming restrictions of distributed protocols) during normal operation. Moreover, it configures per-node local agents to exchange routing information and build backup paths via a distributed Interior Gateway Protocol (IGP). Upon network failures, local agents use IGP information to quickly re-establish connectivity. In the following, we refer to this architecture as *IGP-as-a-Backup SDN* or *IBSDN*.

After introducing some background in Sec. II, we develop several contributions. First, in Sec. III, we provide details on the IBSDN architecture. Moreover, we describe how the controller and the IGP control-plane co-exist, both during normal operation and upon failures. Second, in Sec. IV, we identify and explain main performance issues of naive IBSDN implementations, which range from software-based forwarding to post-failure path stretch. We then describe a procedure to be run by local agents which significantly reduces or fully avoids those performance issues. Third, in Sec. V, we prove desirable properties of IBSDN. Notably, it provably guarantees maximal robustness, since the IGP enables recovery from any combination of failures in the data-plane (devices and links), even when coupled with control-plane (e.g., controller) failures. This actually makes IBSDN an effective alternative both to (i) implementations of the controller as a distributed system (like [11]), used to improve control-plane robustness; and (ii) more complex control-plane designs (like [12]) aimed at supporting safe controller updates. In addition, we show that possible IBSDN performance issues are successfully removed in a (short) bounded time. Fourth, in Sec. VI, we describe an IBSDN prototype implementation that we used to confirm its ability to avoid packet losses upon failures, while quickly converging to a limited post-failure path stretch. Our prototype demonstrates that IBSDN can be implemented in current OpenFlow devices with minimal modifications. Fifth, in Sec. VII, we qualitatively and experimentally compare IBSDN with previous solutions, showing that it overcomes shortcomings of prior work. Finally, we discuss main benefits and limitations of IBSDN in Sec. VIII, and we conclude in Sec. IX.

II. Background

A communication network basically consists of intermediate systems (switches, routers, etc.), which we refer to as *nodes*, and interconnections or *links* between them. Nodes are responsible to forward user traffic. Depending on the traffic flow to which they belong, user packets are generally forwarded through different sequences of nodes, i.e., *forwarding paths*. To forward packets, nodes rely on a data structure called Forwarding Information Base (*FIB*).

Traditionally, distributed routing protocols are used by each network node to compute forwarding paths. In particular, routing protocols enable nodes to exchange the information needed to compute the best forwarding path according to some (configurable) metric. IGPs are the protocols employed for intra-domain routing, i.e., to compute forwarding paths between each source and each destination in the same network. In their basic configuration, IGPs implement simple abstractions. For example, link-state IGPs enable nodes to exchange the network topology, i.e., a map of the network where links are weighted, so that each node computes forwarding paths running the Dijkstra algorithm on the shared topology. In this paper, we restrict to link-state IGPs as they are widely used.

By definition, IGPs do not allow the computation of arbitrary forwarding paths, e.g., hampering minimization of resource consumption and realization of source- or application-based policies. Moreover, because of their distributed nature, configuring routing protocols and fine-tuning their behavior are difficult management tasks for network operators. One of the promises of the SDN proposal is to overcome those difficulties by logic centralization of forwarding path computation.

For this purpose, many SDN proposals are based on the OpenFlow (OF) protocol. Contrary to IGPs, OF provides a set of commands for external software to read and write FIB entries of the nodes. This way, it enables to physically separate packet forwarding (i.e., data-plane actions) from end-to-end path computation (i.e., control-plane decisions), with the former function kept on the nodes and the latter moved to a centralized controller. Moreover, OF rules can match any field in traversing packets and perform a wide set of actions (among which packet field rewriting) on them, while IGP only controls the next-hop on computed forwarding paths and only implements destination-based matching.

Conceptually, IGP and OF realizes heterogeneous approaches to compute and install forwarding paths. Consequently, they have very different features. IGPs are good at finding available paths in a network, and recompute them dynamically when the topology changes. OF simplifies the implementation of complex policies for the selection of very specific and optimized paths, e.g., dynamically mapping new flows to less congested paths. In general, both those abilities are needed. In the following, we show how IBSDN profitably combines IGP and OF to solve different networking problems.

III. IBSDN Basics

In this section, we describe the IBSDN architecture (Sec. III-A), and we illustrate its basic operation, detailing the interaction between its control- and data-plane (Sec. III-B).

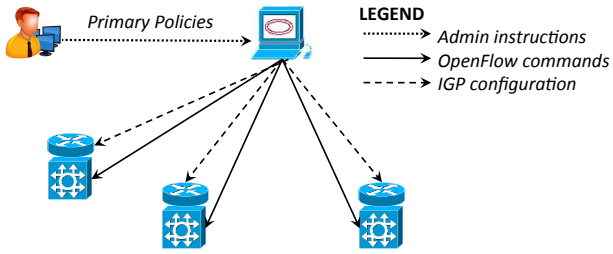


Fig. 1. Architecture of an IBSDN network.

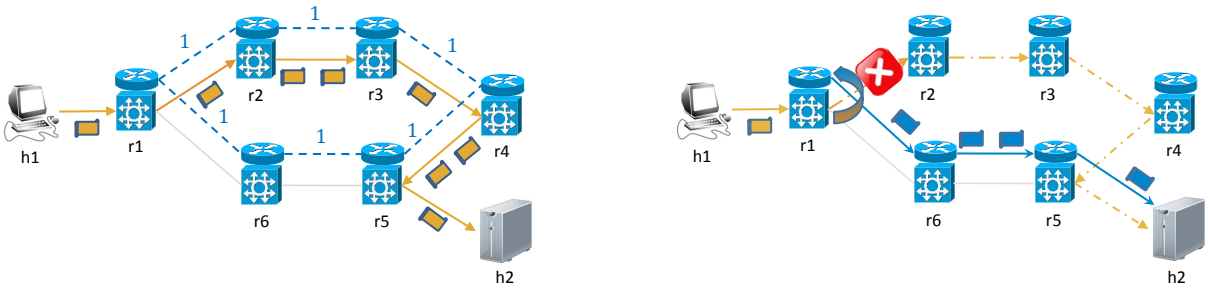
A. IBSDN architecture

The high-level architecture of an IBSDN network is depicted in Fig. 1. Its main components are the IBSDN controller, network nodes, and per-node local agents. In the figure, the IBSDN controller is represented by the computer icon on the top middle of the figure, network nodes by the square icons in the bottom part of the figure, and local agents by router icons above each node. Node and local agent icons are one on top of the other to convey the information that local agents act as routing daemons on top of each OF-capable nodes. Finally, the IBSDN network administrator is represented by the human icon on the top left of the figure. Links between nodes are omitted in the figure for readability.

The main interactions between IBSDN components are also shown on Fig. 1. The administrator specifies to the IBSDN controller the policies to be enforced during normal network operation, i.e., in the absence of failures. We refer to those policies as *primary policies*. Given a set of primary policies, the IBSDN controller computes compliant forwarding paths, and uses OF to install them on the nodes. Since they reflect primary policies, we refer to OF-controlled FIB entries as *OF rules* or *primary rules*. Note that the usage of OpenFlow allows for arbitrary forwarding behavior during normal operation (e.g., source-based or non shortest-path routing). Hence, IBSDN normal-operation capabilities (e.g., for traffic engineering) exceed those of IGP-based solutions (see, e.g., [13]).

Moreover, the controller instructs all local agents to run a link-state IGP (e.g., OSPF or IS-IS). By relying on a minimal IGP configuration (even just defining IGP links and weights), IGP forwarding paths for every pair of network nodes are then computed by local agents, independently from the IBSDN controller. However, those paths are not straightforwardly reflected in the FIB of any node. Rather, routing information is kept in the local agent software memory, and used only when a failure occurs. For this reason, we indicate IGP information as *backup rules*. Also, we refer to packets forwarded using IGP information as *IGP-forwarded packets*, and we assume that IGP-forwarded packets are univocally identifiable, e.g., through a specific tag which we call *IGP tag*.

In addition to (primary and backup) forwarding rules, the IBSDN controller always provides each node with a group of special OpenFlow rules which we call *control rules*. In particular, pre-installed control rules consists of (i) *next-hop control rules*, i.e., a set of rules that are applied if the next-hop on a given port is not reachable (e.g., in the case of link or node failure); and (ii) *IGP-path control rule*, i.e., a rule which matches IGP-forwarded packet. In both cases, the control rules instruct the node to use IGP information to forward packets.



(a) Normal operation

(b) Reaction to a failure

Fig. 2. IBSDN enables packet rerouting without involving the network controller.

Note that the number of control rules, specifically needed in the IBSDN approach, is very limited, since one next-hop control rule per port and one IGP-path control rule in total needs to be added to each node. We provide more details on how to specify those rules with the current OpenFlow protocol in Sec. VI.

B. IBSDN operation

During normal operation, only primarily rules are directly installed in the node FIBs, and traffic flows over OF-defined forwarding paths. Assuming the correctness of the IBSDN controller, this directly implies that primary policies are enforced. Conversely, to avoid traffic losses in the case of failures, flows traversing the disrupted OpenFlow paths are partially moved to backup paths, built using backup IGP rules. Thanks to the control rules configured on each node, the transition from normal operation paths to backup ones does not need involvement of the IBSDN controller and it is virtually lossless. Indeed, the nodes adjacent to a failed link or node can locally detect the failure, e.g., by relying on liveness mechanisms like [14]. As soon as a failure is detected, directly impacted nodes start to use their respective next-hop control rules, and to re-reroute incoming packets that primary rules would send to the failed resource, according to IGP rules. By matching IGP-path control rules on the other routers, those packets then follow the IGP path until their destination.

Fig. 2 illustrates failure reaction in IBSDN through a simple example. The same graphical convention of Fig. 1 is adopted to indicate nodes and local agents. The network administrator and the IBSDN controller are not included in the figure, because they play no active role in the reaction to failures. In addition, (orange) arrows represent OF rules. Dashed (blue) segments between local agents and numbers next to them respectively depict IGP links established by local agents, and their respective weight. Globally, they represent the IGP configuration installed by the IBSDN controller on the local agents. Such configuration generates the backup IGP rules. Light (gray) solid segments between nodes, like the one between $r1$ and $r6$, represent physical links not used by OF.

In particular, Fig. 2 shows the forwarding paths from host $h1$ to host $h2$ before and after a link failure. Packets are represented in the figure by small filled rectangles, and the forwarding path followed by those packets is represented by the sequence of solid arrows. Note that the internal color of the rectangles indicates the rules (i.e., primary or backup ones) used for the corresponding packet forwarding rather than a modification of the packets themselves. In Fig. 2(a), all network links are active and the forwarding path is entirely dictated by OpenFlow rules. Hence, the considered traffic flows over path $(h1\ r1\ r2\ r3\ r4\ r5\ h2)$. Assume now that the link between $r1$ and $r2$ suddenly fails, as highlighted by the cross icon in Fig. 2(b). In this case, the primary path is disrupted, as highlighted by the dashed and dotted orange arrows in Fig. 2(b). As soon as $r1$ detects the failure, incoming packets match the $r1$'s next-hop control rule relative to its $r2$-facing interface. This rule induces $r1$ to locally use its IGP forwarding entry instead of the original OF one, hence making $r1$ avoid the disrupted OpenFlow path. Consequently, the traffic from $h1$ to $h2$ is re-routed over the $(h1\ r1\ r6\ r5\ h2)$ backup path.

IV. Ensuring Post-Failure Forwarding Efficiency

This section describes the techniques that we propose to ensure efficient forwarding over post-failure paths. In particular, Sec. IV-A describes efficiency issues of a naive IBSDN realization, and Sec. IV-B presents our proposal to solve them.

A. Inefficiencies in vanilla IBSDN

In the naive IBSDN implementation described in Sec. III, packets (partially) following IGP paths are not forwarded at line-card speed. Indeed, IGP routes are stored by local agents, i.e., in software. This choice is deliberately made not to increase the size of node FIBs, which are critical and expensive resources. However, it involves that IGP-forwarded packets are inefficiently processed in software on network nodes. In the following, we refer to this issue as *slow local forwarding*.

To visualize a slow local forwarding case, consider the example in Fig. 2. In the failure reaction illustrated in Fig. 2(b), $r1$ applies the next-hop control rule to locally re-route packets according to the IGP. Hence, before being forwarded to the next-hop of $r1$, the packets need to be internally processed by the $r1$'s local agent. This can significantly impact the time taken by packet forwarding as well as node CPU usage and local agent scalability. Moreover, although $r1$ is the only router applying a local re-route, $r6$ and $r5$ also need to process packets in software, because of the IGP-path control rule. In other terms, the slow local forwarding applies to the entire part of the post-failure forwarding path managed by IGP.

Forwarding inefficiency over post-failure paths is even exacerbated by the fact that physical failures can cause packets to bounce back to already traversed nodes. In the following, we refer those packet bounces as *packet returns*.

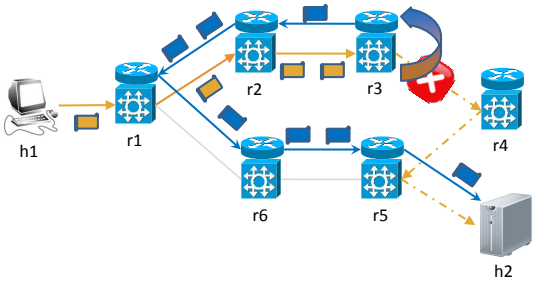


Fig. 3. Post-failure paths can incur significant stretch in vanilla IBSDN.

1: ***process_igp_packet***$(pkt, tags2dests)$
2: **if** pkt is not destined to this router **then**
3: tag ← extract_matching_tag(pkt)
4: installed_rule ← get_OF_rule(matching_tag=tag)
5: dest ← tags2dests(tag)
6: igp_nhs ← get_IGP_rib(dest)
7: new_rule ← create_OF_rule(matching_tag=tag,set IGP_tag && fwd to igp_nhs)
8: change_OF_rule(rule,new_rule)
9: **end if**

Fig. 4. Local packet return removal algorithm implemented by local agents.

Consider again the IBSDN network example in Fig. 2(a), and assume that the link between $r3$ and $r4$ fails, as in Fig. 3. Referring again to the flow of packets from $h1$ to $h2$, the first node that can detect the given link failure is $r3$. It then plays the same role as $r1$ in Fig. 2(b): Upon failure detection, $r3$ starts using a next-hop control rule, and locally forwarding according to IGP information. In turn, IGP automatically recomputes the only possible path from $r3$ to $h2$ in the new topology. As a result, data packets are forwarded from $r3$ back to $r2$. By matching its IGP-path control rule, $r2$ also uses its own IGP next-hop. Ultimately, packets of the considered flow follow the forwarding path $(h1\ r1\ r2\ r3\ r2\ r1\ r6\ r5\ h2)$. That is, packet returns occur at $r2$ and $r1$, as those nodes receives back (from $r3$ and $r2$ resp.) IGP-forwarded packets already processed according to primary rules.

Two relevant observations should be immediately made on packet returns. First, they never disrupt connectivity. Indeed, thanks to the IGP-path control rule installed on all nodes, *packet returns never cause any packet to loop indefinitely in IBSDN*. A proof of this statement is reported in Sec. V. Second, packet returns causes post-failure path stretch. In Fig. 3, for example, the $h1-h2$ flow unnecessarily traverses links $(r1, r2)$ and $(r2, r3)$ in both directions, stretching the post-failure forwarding path by 4 hops. This unfortunately can induce performance degradation, e.g., non-negligibly increasing delay, especially when combined with slow local forwarding.

B. Efficient IBSDN with local computation

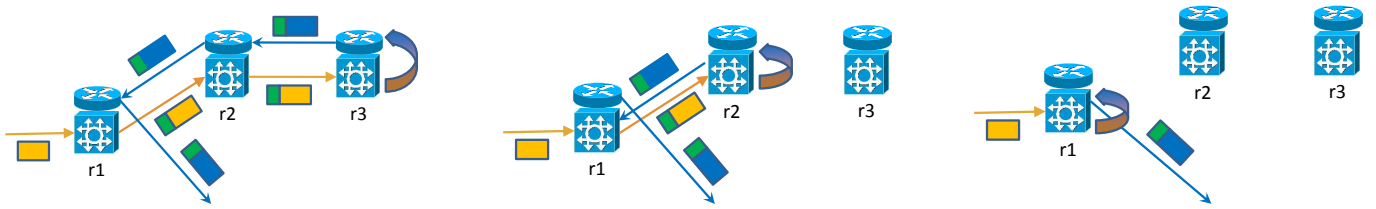
To overcome IBSDN potential inefficiencies, we rely on a *packet return removal procedure*. This procedure is based on providing local agents with enough intelligence to locally optimize forwarding decisions. This means that packet returns can be efficiently removed *without the need for any coordination with the IBSDN controller*.

The packet return removal procedure is based on a FIB update algorithm triggered by data-plane packets. Indeed, the reception of an IGP-forwarded packet can be interpreted by any node as an implicit notification of a network failure. Local agents can then deduce that the IGP has to be used for the incoming flow to which the IGP-forwarded packets belong. Hence, they can update the FIB of their corresponding nodes by replacing the primary rule matching the affected flow with a new OF rule reflecting the IGP information. To avoid inconsistent forwarding on other nodes, the new OF rule must identify the matching packets as IGP-forwarded.

Unfortunately, the OpenFlow rule to be updated is not straightforward to identify from an IGP-forwarded packet. Indeed, OpenFlow rules can match any field in the packet. Hence, the destination-based lookup implemented by IGP does not guarantee the correct identification of the rule used by OF for the same packet. As a workaround, local agents may simulate the packet matching process performed by nodes. Beyond being time and space consuming, this workaround is not even correct if packets are modified en route along primary paths, e.g., to support advanced features like middleboxing [15].

To enable correct and fast identification of the rule to be updated, we propose that the IBSDN implementation relies on a specific packet tagging technique. Namely, the controller configures ingress nodes of each flow to add a special tag to each forwarded packets. This tag, which we call *ingress tag*, is used by all network nodes to match packets, and must be never modified by traversed nodes (except the egress node, which can remove it). The role of the ingress tag is to uniquely identify the treatment (forwarding path, possible field rewriting, etc.) to be applied to incoming packets. During normal operation, ingress tags can be used to accommodate primary policies in an efficient way. That is, the IBSDN controller can configure ingress nodes to apply the same ingress tag to packets belonging to multiple flows if they require similar processing (i.e., to follow the same forwarding path and undergo the same field changes). This would allow FIB simplification and compression on the internal nodes. Moreover, since they are never overwritten along the forwarding path, ingress tags also enable easy identification of the rule to be updated when an IGP-forwarded packet is received, irrespectively of possible modifications previously applied to it.

The full algorithm that is run by local agents within the packet return removal procedure in the presence of ingress tags is reported in Fig. 4. We call it *local packet return removal algorithm*. The algorithm takes as input a traversing data packet and a mapping between ingress tags and network destinations. Since ingress tags are computed by the IBSDN controller, this mapping can be provided to local agents by the controller itself. Since the algorithm is run by local agents, it does not apply to packets forwarded on primary paths. Moreover, no action is taken for packets directed to the node itself, like IGP control-plane packets (line 2). For all the other packets (i.e., IGP-forwarded packets for a remote destination), the algorithm extracts the ingress tag (line 3), and relies on it to determine both the OF rule to be replaced and the IGP next-hops to be used (lines 4-6). Based on this information, a new OF rule is computed (line 7) and pushed to the node FIB (line 8). The algorithm can be easily implemented in an efficient way, as both primary (OF) and backup (IGP) rules can be represented by associative maps with constant time access.



(a) Traffic forwarding immediately after the failure shown in Fig. 3

(b) Traffic forwarding after $r2$ runs the local packet return removal procedure (Fig. 4)

(c) Traffic forwarding after $r1$ runs the local packet return removal procedure (Fig. 4)

Fig. 5. Application of the packet return removal procedure to the example in Fig. 3.

Note however that, after updating a given FIB entry, any local agent must keep it synchronized with IGP information, e.g., to ensure that the final IGP next-hop is actually used if some time is needed for IGP to converge to the final backup path.

Fig. 5 shows an example of application of the packet return removal procedure when local agents implement Fig. 4 algorithm. The example refers to the case depicted in Fig. 3. Data packets are represented by filled rectangles, split in two when an ingress tag is present on them. Note that $r1$ adds a green ingress tag to the beginning of traversing packets, since it is the ingress point for the considered $h1-h2$ flow. The ingress tag is not modified by any other node. The packet returns raised by the unavailability of the primary path at $r3$ and reported in Fig. 5(a) are removed as soon as the first IGP-forwarded packet is processed by $r1$. Few packets, traversing $r1$ before it completes such an algorithm, can follow the path illustrated in Fig. 5(b). The final state is depicted in Fig. 5(c). Solid arrows representing backup rules (i.e., between $r2$ and $r1$ and from $r1$, resp.) highlights in Figs. 5(b) and 5(c) that slow local forwarding issues are quickly solved on $r2$ and $r1$ respectively. Since they run the packet return removal procedure, nodes between $r1$ and $h2$ also update their FIB entries according to IGP. In other words, a good side effect of our packet return removal procedure is to avoid slow local forwarding.

Generally speaking, our packet return removal procedure has several desirable properties. First, it removes any packet return in a finite time proportional to its length. Second, it quickly avoids slow local forwarding. Note that rapidly solving efficiency problems mitigates the risk of congesting local agents with line-speed forwarded traffic. Nevertheless, such a risk can be completely avoided only through an appropriate node implementation, e.g., limiting the number of packet redirected to local agents (at the cost of re-introducing some packet loss) or ensuring fast FIB updates. Finally, the packet return removal procedure preserves packet delivery guarantees. Formal proofs of those properties are reported in Sec. V.

V. IBSDN Guarantees

To prove IBSDN properties, we introduce few notation. We denote the forwarding path at time t between a source s and a destination d as $\pi_t(s,d)$. Accordingly, we define a packet return L as a sub-path of some forwarding path $\pi(s,d)$, such that $L = (v_0 \dots v_k\ v_0)$, with $k \geq 1$.

We now show that IBSDN is safe, guarantees maximal robustness and efficiently avoids previously-described forwarding inefficiencies. In the following, we refer to the complete realization of IBSDN including the packet return removal procedure defined in Sec. IV. Vanilla IBSDN can be considered as a special case, for which performance guarantees do not hold. In both cases, IBSDN features the following properties.

Property 1: Nodes adjacent to a failure rely on IGP to re-route packets from primary paths disrupted by the failure.

Property 2: Packets from any source s to any destination d are tagged as IGP-forwarded by the first node $x \in \pi(s,d)$ that relies on IGP information to forward packets.

Property 3: Any node forwards any IGP-forwarded packet that it receives to its IGP next-hop.

Properties 1 and 2 are ensured by the next-hop control rules, while Property 3 holds by definition of the IGP-path control rules. Note that the validity of the properties is not impacted by the packet return removal procedure since packets forwarded using IGP are also tagged as IGP-forwarded by the OF rules installed by the procedure (see line 7 in Fig. 4).

Those properties ensures the following theorem.

Theorem 1: In IBSDN, connectivity is re-established upon any combination of failures that does not partition the physical network, without any action of the controller.

Proof: Consider an IBSDN network subject to any combination of failures that does not induce network partitions. Since all local agents run an IGP, they also compute a new IGP path for each source-destination pair. Let t be any time after new IGP path computation terminates on local agents. We now prove that at t, packets are delivered from any source s to any destination d. Consider any primary path $P = (s \dots d)$. If P contains no failed link, only primary rules are used along P by definition of IBSDN and the statement directly follows. Otherwise, let (x,y) be a failed link such that $P = (s \dots x\ y \dots d)$ and no other failed links exist in the sub-path of P between s and x. We define $r \in P$ as the node, closest to s in P, that uses IGP to forward packets from s to d. Node r must exist, since Property 1 guarantees that at least one node, i.e., x, would use IGP if it receives packets from s to d. By Properties 2 and 3, the IGP path is then followed from r until d, which proves the statement. ■

Corollary 1: Packets never loop indefinitely in IBSDN.

Corollary 2: IBSDN ensures maximal robustness.

In addition, packet returns are removed and slow local forwarding is avoided in a short time after failures. To provide an upper bound of the time needed for those operations, we define the *delay* $\delta(P)$ *over a path* $P = (v_0 \dots v_k)$ as the sum of the delays introduced by each node $v_i \in P$, e.g., to

process packets, and each link $(v_i\ v_{i+1})$ traversed by P, e.g., to transmit packets on the link. In particular, the delay $\delta(L)$ over a packet return L is the sum of the processing delay on each node and link belonging to L. For the sake of simplicity, we assume that IGP has already converged on post-failure paths.

Theorem 2: IBSDN successfully removes any packet return L in a time which is bounded by the delay over L.

Proof: Consider an IBSDN network subject to an arbitrary set of failures. Let L be a packet return generated at time t_0 and affecting traffic from any source s to any destination d. By definition of packet return, $\pi_{t_0}(s,d)$ must traverse all the nodes of the packet return. Let $x \in L$ be the node such that the sub-path of $\pi_{t_0}(s,d)$ from s to x contains no other node in L. That is, $\pi_{t_0}(s,d) = (s \dots l_0 \dots l_k\ l_0\ f \dots d)$, where $l_i \in L$ $\forall i = 0, \dots, k$, $l_0 = x$, and f is the IGP next-hop of x. By Theorem 1, packets traverse all the nodes in $\pi_{t_0}(s,d)$ and reach d. This means that packets forwarded over $\pi(s,d)$ will traverse x twice, the second time being tagged as IGP. By Alg. 4, after receiving the first IGP-forwarded packet of the $s-d$ flow, x will update its FIB to directly send packets of this flow to f. Let t_1 and t_2 be the times at which x respectively receives the first packet of the $s-d$ flow and updates the corresponding FIB entry. Then, $\pi_{t_2}(s,d) = (s \dots l_0\ f \dots d)$ does not contain the L packet return. The statement follows by noting that x will update its next-hop as soon as it receives the first IGP-forwarded packet, that is, at time $t_2 = t_1 + \delta(L)$. ■

Theorem 3: IBSDN removes slow local forwarding in a time which is bounded by the delay over the network path with the maximum number of nodes.

Proof: Consider an IBSDN network subject to an arbitrary set of failures, and let $s-d$ be a source-destination pair such that the primary path between them is disrupted by such failures. Upon the failures, at time t_0, $\pi_{t_0}(s,d)$ must be a concatenation of two paths P and Q, where P (resp. Q) is determined by primary (resp. backup) rules. By Theorem 1, packets traverse all the nodes in $\pi_{t_0}(s,d)$ and reach d. Moreover, each node on Q suffers the slow local forwarding issue at t_0. By definition of Alg. 4, each time a node in Q receives an IGP-forwarded packet, it updates the OF rule that applies to the $s-d$ flow. Hence, when the first packet of the flow sent after t_0 is delivered to d, all the nodes in Q have updated their FIB. This directly yields the statement. ■

VI. Implementation and Evaluation

This section describes the implementation and evaluation of IBSDN. Sec. VI-A overviews our running prototype. We used this prototype to evaluate IBSDN in two settings, namely, (i) a *micro-benchmark* consisting of a small synthetic topology designed to explore corner cases; and (ii) a *macro-benchmark* targeted to evaluate the efficiency of IBSDN (with and without the packet returns removal procedure) in realistic networks. We focused on single link failures as they are the most frequent failures in large networks [16]. Sec. VI-B and Sec. VI-C respectively discuss micro- and macro-benchmark results.

A. IBSDN prototype

We prototyped all components of the IBSDN architecture shown in Fig. 1. Namely, we implemented IBSDN nodes as Linux hosts whose forwarding is controlled by Open vSwitch (OVS) [17]. We relied on the BIRD routing daemon[1] for local agent realization. The IBSDN controller is based on the Ryu[2] framework. However, we extended the latter framework so that the controller both installs OF rules on the nodes by communicating with OVS and configures local agents to run OSPF. Finally, we relied on User Mode Linux to create virtual IBSDN networks.

In our prototype, we implemented IGP tags and ingress tags, respectively needed for identifying IGP-forwarded packets and flows to which packets belong (see Sec. III), as follows. The IGP tag corresponds to a specific value (i.e., 32) of the type of service (`tos`) IP header field. This value has to be considered as reserved in our IBSDN prototype, i.e., it cannot be used for other purposes. Moreover, since OF rules match only the IP destination of traversing packets and nodes do not overwrite any packet field in our experiments, we used destination and source IP addresses and ports as ingress tags. Our prototype exemplifies how IBSDN-specific tags may be defined as unused or unmodified packet header fields.

We used those tags, along with OpenFlow NORMAL port and fast failover groups, to express control rules on each node, and define which packets need to be forwarded to local agents. The NORMAL port represents the traditional non-OpenFlow pipeline of the switch. By default, OVS behaves as a Layer-2 learning switch when packets are sent to the NORMAL port. We patched OVS to push packets to the host's kernel networking stack and be processed according to IGP information, when the NORMAL port is specified as output port. We then implemented the IGP-path control rule by sending data packets to the NORMAL port when matching the IGP tag. To enable IGP communication between local agents on different nodes, we also used the NORMAL port as output for IGP control-plane messages. Fast failover groups enable to specify the rule to be used in the presence of a failure on a node interface. Hence, for each node, next-hop control rules are implemented configuring one fast failover group for each interface and specifying the NORMAL port as output of the rule applied in the case of failure on that interface.

Our prototype shows that IBSDN can be implemented with minimally-modified switches compliant with any OpenFlow standard version higher than or equal to 1.1.0 [4].

B. Micro-benchmarking IBSDN effectiveness

We run our IBSDN prototype on a simplistic network topology. To model corner cases for IBSDN, we consider a topology similar to the one depicted in Fig. 2(a), except for the addition of vertical links between $r2$ and $r6$, and between $r3$ and $r5$. We add those links to create more paths between $h1$ and $h2$, and verify that IBSDN post-failure paths coincide with the shortest ones in the post-failure topology.

We perform multiple experiments, varying the number of $h1-h2$ flows between 10 and $1,000$. In each of those experiments, we simulate the failure of the penultimate link in the primary path, that is, $(r4, r5)$ in Fig. 2(a). The failure impacts all the flows, as shown in Fig. 2(a). Since we configured

[1] see http://bird.network.cz/

[2] see http://osrg.github.io/ryu

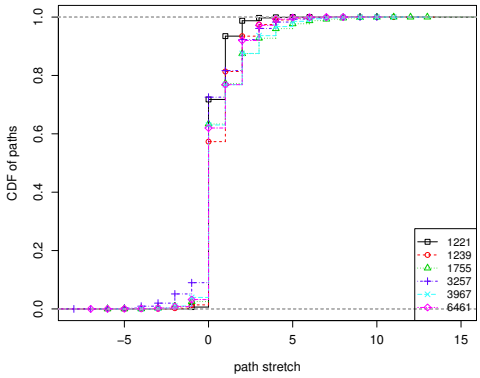


Fig. 6. The number of additional hops (path stretch) of IBSDN paths with respect to the corresponding IGP paths is likely limited on realistic networks.

vertical links with high IGP weights, the simulated failures cause IBSDN to react as in Fig. 3. In each experiment, we also monitor forwarding paths and packet losses by sending probe packets of each flow every 0.05 seconds.

Our experiments confirmed that IBSDN would lead to **no packet loss** in our micro-benchmark setting, independently of the number of configured flows. This is partially due to the fact that the IGP is already converged to its post-failure paths (because of the IGP weights), when the failure occurs. Indeed, IGP convergence is known to be a potential source of packet losses [18]. However, commercial devices are reported to complete the IGP convergence in sub-seconds, even in very large networks [10]. Moreover, fast re-route techniques (e.g., [19]) can be used to avoid IGP convergence problems.

C. Macro-benchmarking IBSDN efficiency

To evaluate the efficiency of IBSDN on realistic large networks, we simulate single link failures in the Rocketfuel topologies[3] [20]. We considered the case in which a forwarding path needs to be installed for any source-destination pair in the network. For the sake of simplicity, primary and backup paths are both computed as the shortest paths, on the original and post-failure topologies respectively.

Fig. 6 shows that the path stretch is very limited in the vast majority of our experiments on all Rocketfuel topologies. In particular, the plot contains one curve per topology, and each curve represents the cumulative distribution (CDF) of IBSDN paths having at most a given stretch. The stretch is calculated as the difference in the number of hops between the IBSDN path (after packet return removal) and the post-failure IGP path (which is the shortest path in the new topology). For all the Rocketfuel topologies, the stretch is 0 for more than 50% of the post-failure paths, and it is at most 1 in more than 70% of the cases. In addition, the 95-th percentile of the path stretches respectively ranges from 2 (in AS 1221) to 4 (in ASes 1755 and 3967), and the 98-th percentile varies between 2 (in AS 1221) and 6 (in AS 1755). The remaining outliers are so few that static backup entries can be used for them (if the corresponding stretch absolutely needs to be avoided), i.e., as an exception to the IBSDN behavior. Observe that the path stretch has a negative value in few cases, in which a primary path brings the packets up to a node which is very close to the destination in the post-failure topology.

The limited path stretch induced by IBSDN is partially due to our packet return removal procedure. Across all experiments, this procedure eliminates at least one packet return in a number of post-failure paths ranging from about 10% to about 25%. Without this procedure, packet returns would have contributed to increase path stretches by values varying between 2 and 13. Note that the majority of removed packet returns have limited size, and are quickly eliminated by the packet return removal procedure (see Theorem 2).

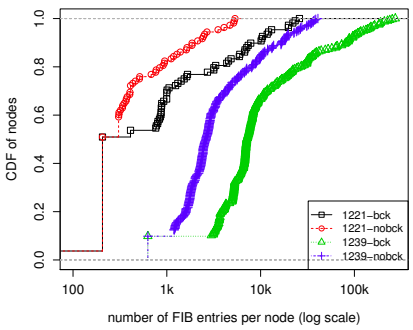
[3]publicly available at http://www.cs.washington.edu/research/projects/networking/www/rocketfuel

VII. Comparison with Related Work

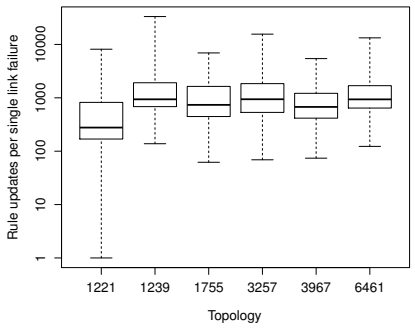
Previous approaches to react to failures in SDN networks can be classified in two main categories: proactive and reactive.

Proactive approaches, like [7], have very high hardware requirements, since they are based on pre-installing backup entries on each node for each failure. This asks for the availability of a huge amount of FIB entries on each node if connectivity has to be preserved for any failure. Indeed, the number of additional FIB entries needed by proactive approaches is proportional to the number of both installed flows and the number of possible failures. As an illustration, Fig. 7(a) plots as CDFs the number of FIB entries per node needed in the proactive approach versus those required by IBSDN to ensure robustness of Rocketfuel topologies for any single link failure. We computed primary paths as the shortest paths in the original topologies. For plot readability, we only show the best (i.e., 1221) and the worst (i.e., 1239) case in terms of backup entries per node. The plot highlights that the proactive approach generally needs one order of magnitude more FIB entries than IBSDN.

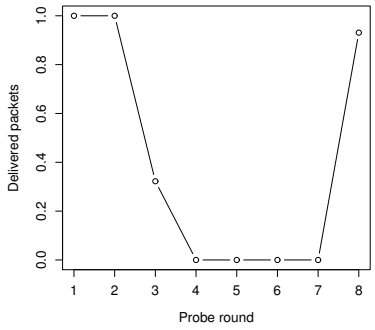
Reactive approaches, like [9], fully rely on the centralized controller to update forwarding paths in the case of failure. That is, when a failure occurs, a special message is sent to the controller, which has to identify the affected flows, recalculate feasible (or optimal) forwarding paths, and update the FIB of the impacted nodes accordingly. To study the overhead and control-plane scalability challenges posed by this approach, we simulated single link failures on Rocketfuel topologies, with shortest routing used to compute one primary and one backup path per node pair. Fig. 7(b) plots the number of OF rule updates to be pushed by a centralized controller in the reactive approach. Whiskers represent maximum and minimum values, the borders of each box correspond to the 75-th and the 25-th percentiles, and the thick lines in the boxes identify median values. For all topologies, the median of OF rule updates is between 100 and $1,000$, with maximum values of more than $30,000$ rule updates in the worst case (for AS 1239). Even if the updates are pre-computed and quickly retrieved by the controller, transferring and installing them to nodes (possibly while guaranteeing the absence of transient problems [21]) requires an extremely performant connection between controller and nodes, creates spikes of control message overhead, and possibly overloads the controller during failure recovery. Moreover, if a reactive approach is adopted, failures of the connection between the controller and the nodes or of the controller itself make the network unable to react to



(a) CDF of the number of FIB entries needed in the proactive approach (`bck`) versus the IBSDN and reactive (`nobck`) ones.



(b) Number of FIB entries to be updated (e.g., by a reactive SDN controller) in case of single link failures.



(c) Contrary to IBSDN, reactive approaches show non-negligible packet losses after a failure in our micro-benchmark topology.

Fig. 7. Our experiments expose intrinsic limitations of approaches alternative to IBSDN.

any other event. In contrast, the number of rule updates shown in Fig. 7(b) is practically split among local agents in IBSDN, and no involvement is needed from the controller.

We also used our micro-benchmark to estimate the impact of such a number of rule updates on user traffic in the presence of a non-negligible delay in the connection between the controller and nodes (as in geographically-distributed networks). Fig. 7(c) shows the results of an experiment in which i) a single link fails, ii) a single node has to be updated to reroute $1,000$ traversing flows, iii) the controller pre-computes OF rule updates, iv) the connection between the controller and the nodes induces a delay of 100 milliseconds, and v) packet probes are sent on all impacted flows, in *rounds*, every 0.05 seconds. The plot highlights that the reactive approach can produce significant disconnections, even in the best-case scenario (small topology, rule update pre-computation, single node to updated, etc.) that we considered. It also underlines the criticality of the controller-node connections for the reactive approach. Also, it suggest the difficulty to reroute traffic in an arbitrary short time. By relying on the distributed IGP for failure reaction, IBSDN exhibits no packet loss in the micro-benchmark scenario, and can rely on IGP fast reroute techniques to avoid losses in the general case. Note that IBSDN holds the same advantages also with respect to reactive approaches of IGP-based autonomic management systems [22].

Contrary to IBSDN, problems of proactive and reactive approaches are exacerbated if additional failure scenarios (e.g., single node or multiple link failures, as for Shared Risk Link Groups) are considered. This also hampers the applicability of variations of prior approaches. For example, the practicality of proactive approaches relying on a software FIB are still limited by the number of needed backup entries.

To avoid intrinsic limitations of OF-based solutions, IBSDN leverages the presence of two control-planes, implemented by the controller and IGP-running local agents respectively. The simultaneous usage of IGP and OpenFlow has also been proposed in [23]. Contrary to [23], IBSDN runs the IGP directly on the nodes (rather than just on the controller to compute forwarding paths), hence benefiting from IGP failure recovery abilities. In the context of traditional networks, the idea of relying on multiple control-planes running different instances of a distributed protocol has also been exploited by few other works, for purposes ranging from fast failure recovery [24] to pre-deployment configuration evaluation [25] and network reconfigurations [26], [27]. IBSDN uses multiple control-planes to combine SDN and IGP, introducing tailored techniques to efficiently ensure network robustness. According to [6], IBSDN is a service-based hybrid SDN architecture. As such, beyond possibly being a desirable long-term design (as we argued in this paper), it can also provide incentives and enable specific strategies to transition to SDN.

VIII. Discussion

IBSDN splits the problem of computing forwarding paths into (i) a long-term optimization problem, tackled by a centralized controller, and (ii) a short-term feasibility problem, in which a distributed algorithm ensures connectivity in the case of failures. This problem breakdown has several consequences.

On one hand, the usage of distributed algorithms for connectivity preservation has multiple advantages. First, it allows a clear separation between optimization (traffic engineering, delay minimization, etc.) and robustness goals, and the solution of the corresponding sub-problems with the networking paradigm (centralized or distributed) which is respectively more suitable. This separation leads to a simplification of both the controller (which can rely on an IGP module for robustness) and the IGP configuration (since advanced features like traffic engineering extensions and careful weight tuning are unnecessary). It also makes the SDN controller more scalable, as it does not have to directly manage many (possibly transient) events. Note that the peculiar usage of IGP made by IBSDN avoids known problems, e.g., due to the presence of BGP [28], affecting internal routing re-optimization in traditional networks. Second, IBSDN ensures maximal robustness (see Sec. V) with an extremely limited number of additional FIB entries (i.e., for the control rules) per router. Previous IGP work (e.g., [19]) can be leveraged to transform IBSDN robustness into a proven ability to quickly reroute traffic. Third, IBSDN minimizes the impact of controller failures, and provide operators with means to debug and operate the network independently of controller misbehaviors.

On the other hand, relying on IGP for failure recovery has some intrinsic limitations. Primarily, IGP has not the same expressive power as SDN protocols like OpenFlow. For example, IGP is limited to destination-based forwarding. Hence, backup paths compliant with primary policies may

not be enforced. Moreover, depending on the combination of occurring failures, IGP may need to converge. This process is not harmless in general [18]. However, it has been shown that IGP convergence is fast in real, even large networks [10] and its negative side effects can be strongly mitigated by relying on fast reroute techniques. Third, IBSDN requires some form of coordination between the controller and local agents, especially to avoid unsafe overwriting of OF rules on nodes (e.g., when the usage of the controller has to be restored). For this purpose, the IBSDN controller should always have higher writing priority to the FIB with respect to local agents. This way, the controller would be responsible for overwriting the FIB entries installed by local agents for failure recovery with new entries for post-failure normal operation. Finally, by exchanging IGP messages, local agents also create control-plane communication overhead. However, the wide deployment of IGP testify how this is practically not a major issue.

Globally, those limitations can make IBSDN unsuitable in some cases. However, ensuring basic connectivity (at least, for the majority of flows) upon failures is a classic network requirement. IBSDN facilitates to achieve this goal within an SDN-based architecture. For the cases in which IBSDN does not totally fit, our proposal can be combined with previously-proposed reactive or proactive approaches, while improving their scalability, e.g., by restricting the pre-installed OF rules to the few most critical flows with peculiar requirements.

IX. Conclusions

In this paper, we presented IBSDN, a hybrid SDN architecture in which long-term optimization goals (e.g., for fine-grained traffic engineering) are assigned to a centralized SDN controller while a distributed IGP is used for short-term forwarding path computation (e.g., for failure reaction). Within our proposal, we described techniques to reroute packets, defined mechanisms that ensure data-plane performance when failures occur, and proved maximal robustness and efficiency of IBSDN. By means of a prototype implementation and simulations on realistic topologies, we confirmed the practicality and effectiveness of our architecture, especially when compared with alternative OF-based approaches.

Our results suggest that a deeper investigation of the spectrum between completely centralized (e.g., OF-based) and completely distributed (e.g., IGP) solutions can be an interesting research direction. By improving SDN robustness, IBSDN exemplifies how centralized and decentralized approaches can mutually benefit from being used simultaneously, for different networking problems. In future work, we plan to bring this argument one step further, by studying which combination (e.g., flow-based) of IBSDN, OF-based approaches and IGP techniques leads to the best trade-offs between minimizing FIB entries and ensuring fast packet rerouting.

Acknowledgements

This work has been supported by the ARC grant 13/18-054 from Communauté française de Belgique.

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*, 2010.

[2] 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.

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

[4] Open Networking Foundation, "Openflow switch specification, version 1.1.0," white paper, 2011.

[5] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, "OpFlex Control Protocol," Internet Draft, 2014.

[6] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *Comput. Commun. Rev.*, vol. 44, no. 2, Apr. 2014.

[7] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire: Declarative Fault Tolerance for Software-defined Networks," in *HotSDN*, 2013.

[8] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *HotSDN*, 2013.

[9] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A demonstration of fast failure recovery in software defined networking," in *TRIDENTCOM*, 2012.

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

[11] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *WREN*, 2010.

[12] L. Vanbever, J. Reich, T. Benson, N. Foster, and J. Rexford, "HotSwap: Correct and Efficient Controller Upgrades for Software-defined Networks," in *HotSDN*, 2013.

[13] D. Medhi and K. Ramasamy, *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers Inc., 2007.

[14] D. Katz and D. Ward, "Bidirectional Forwarding Detection (BFD)," RFC 5880, 2010.

[15] Z. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *SIGCOMM*, 2013.

[16] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *Trans. on Netw.*, vol. 16, pp. 749–762, 2008.

[17] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *HotNets*, 2009.

[18] 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. 1280–1932, 2007.

[19] C. Filsfils, P. Francois, M. Shand, B. Decraene, J. Uttaro, N. Leymann, and M. Horneffer, "LFA applicability in SP networks," RFC 6571, 2012.

[20] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *SIGCOMM*, 2002.

[21] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *SIGCOMM*, 2012.

[22] X. Liu, P. Juluri, and D. Medhi, "An experimental study on dynamic network reconfiguration in a virtualized network environment using autonomic management," in *IM*, 2013.

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

[24] A. Kvalbein, A. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP Network Recovery Using Multiple Routing Configurations," in *INFOCOM*, 2006.

[25] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," in *SIGCOMM*, 2008.

[26] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Seamless Network-Wide IGP Migrations," in *SIGCOMM*, 2011.

[27] S. Vissicchio, L. Vanbever, C. Pelsser, L. Cittadini, P. Francois, and O. Bonaventure, "Improving network agility with seamless BGP reconfigurations," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 990–1002, 2013.

[28] 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.