SWIFT: Predictive Fast Reroute

Thomas Holterbach
ETH Zürich; CAIDA, UC San Diego
thomahol@ethz.ch

Alberto Dainotti
CAIDA, UC San Diego
alberto@caida.org

Stefano Vissicchio
University College London
s.vissicchio@cs.ucl.ac.uk

Laurent Vanbever
ETH Zürich
lvanbever@ethz.ch

ABSTRACT

Network operators often face the problem of remote outages in transit networks leading to significant (sometimes on the order of minutes) downtimes. The issue is that BGP, the Internet routing protocol, often converges slowly upon such outages, as large bursts of messages have to be processed and propagated router by router.

In this paper, we present SWIFT, a fast-reroute framework which enables routers to restore connectivity in few seconds upon remote outages. SWIFT is based on two novel techniques. First, SWIFT deals with slow outage notification by predicting the overall extent of a remote failure out of few control-plane (BGP) messages. The key insight is that significant inference speed can be gained at the price of some accuracy. Second, SWIFT introduces a new data-plane encoding scheme, which enables quick and flexible update of the affected forwarding entries. SWIFT is deployable on existing devices, without modifying BGP.

We present a complete implementation of SWIFT and demonstrate that it is both fast and accurate. In our experiments with real BGP traces, SWIFT predicts the extent of a remote outage in few seconds with an accuracy of ~90% and can restore connectivity for 99% of the affected destinations.

CCS CONCEPTS
• Networks → Network performance analysis; Network measurement; Network reliability;

KEYWORDS
BGP; Convergence; Fast Reroute; Root Cause Analysis

1 INTRODUCTION

Many applications nowadays require continuous Internet connectivity, where even the slightest downtime can cause large financial and reputational loss. For example, the cost of one minute of downtime for Amazon or Google easily reaches a 6-digit number [2] and almost any outage that they experience makes the news [4, 8]. Smaller Internet players are not better off. Across the networking industry, the average cost of downtime is estimated to be about $8,000 per minute [55].

Unfortunately, guaranteeing always-on connectivity Internet-wide is a big challenge for network operators. Even if their network is perfectly resilient, they still face the problem of remote outages in transit networks, i.e., connectivity disruptions in external networks forwarding their traffic. These disruptions are frequent: large networks routinely see tens of failures or configuration changes in any single day [12, 26, 28, 35, 65], each potentially disrupting transit traffic for thousands of destinations.

Problem. BGP, the Internet routing protocol, converges slowly upon remote outages. This can result in long data-plane downtime for many destinations, including popular ones. Our measurements on real BGP traces and recent router platforms (§2) show that:

- large bursts of BGP withdrawals (> 1.5k prefixes) regularly happen in the Internet: 53% (resp. 86%) of ~200 BGP sessions distributed worldwide see at least one large burst per week (resp. month). Since single routers in transit networks maintain tens to hundreds of such BGP sessions [27], the probability of receiving a burst is important. 9.5% of these bursts involve more than 20k prefixes, and some involve up to 560k prefixes. Nearly all the biggest (hence slowest) bursts include prefixes for popular destinations (Google, Akamai, Netflix, etc.).

- similarly to what prior studies have shown (e.g., [18, 37, 43, 45, 53, 62, 63, 66]), BGP slow convergence can cause dozens of seconds of data-plane downtime, during which packets towards many destinations are lost. We confirmed data-plane losses with both tested experiments on commercial routers and private conversations with operators.

To further substantiate the problem, we conducted a survey with 72 operators. Our survey indicates that slow BGP convergence is a widespread concern. According to the operators monitoring convergence time (47% of them), BGP takes more than 30 seconds to converge upon remote outages, on average.

Local fast-reroute upon remote outages. We present SWIFT, a fast-reroute framework that enables a router to restore connectivity in few seconds upon remote outages. SWIFT is based on two main...
ingredients. Immediately after receiving the first BGP messages of a burst, a SWIFTED router runs an inference algorithm to localize the outage and predict which prefixes will be affected—a sort of time-bound Root Cause Analysis (RCA). Based on this inference, the SWIFTED router reroutes the potentially affected prefixes on paths unaffected by the inferred failure. As many prefixes may have to be rerouted at once, SWIFT also includes a data-plane encoding scheme that enables the router to flexibly match and reroute all prefixes affected by a remote failure with few data-plane rule updates.

Balancing inference accuracy & speed, with correctness & performance in mind. SWIFT restores connectivity within few seconds by inferring the failure from a single vantage point. This contrasts to prior RCA studies (e.g., [15, 19, 23, 36, 39, 40, 68–70]), which aim at finding causes of outages within minutes, hence can benefit from more flexibility in terms of inference algorithms and input sources (e.g., active probing from multiple vantage points).

The key insight behind SWIFT inference algorithm is that some accuracy can be traded for a significant gain in speed. Identifying the topological region where an outage is happening is indeed much faster than precisely locating the outage within that region. By rerouting traffic around the region, a SWIFTED router immediately restores connectivity for the affected prefixes at the cost of temporarily forwarding few (according to our results) unaffected prefixes on alternate working paths.

SWIFT makes sure that the effect of diverting non-affected traffic does not trump the benefit of saving traffic towards the affected prefixes. First of all, we prove that rerouting non-affected traffic is safe: SWIFT does not lead to forwarding anomalies, even if multiple routers and ASes deploy it. Second, SWIFT selects the alternate paths taking into account the operator’s policies (e.g., type of peers, cost model) and performance criteria (e.g., by preventing to reroute large amount of traffic to low-bandwidth paths).

Deployment. SWIFT is deployable on a per-router basis and does not require cooperation between ASes, nor changes to BGP. SWIFT can be deployed with a simple software update, since the only hardware requirement, a two-stage forwarding table, is readily available in recent router platforms [3].

Whenever a SWIFTED router fast-reroutes upon an outage, it guarantees connectivity to all the traffic sources passing through it. Hence, deploying SWIFT in a few central ASes would benefit the entire Internet, since these ASes would also protect their (non-SWIFTED) customers. The same applies within a network: deploying few SWIFTED routers at the edge boosts convergence network-wide. A full Internet SWIFT deployment would achieve the utmost advantages of our scheme, as it guarantees ASes to reroute quickly, independently, and consistently with their policies.

Performance. We implemented SWIFT and used our implementation to perform extensive experiments using both real and synthetic BGP traces. Across all our experiments, SWIFT correctly identified 90% of the affected prefixes within 2 seconds. Moreover, a SWIFTED router can fast reroute 99% of the predicted prefixes with few data-plane rule updates, i.e., in milliseconds. Finally, we show that our implementation is practical by using it to reduce the convergence time of a recent Cisco router by more than 98%.

Contributions. Our main contributions are:

- A thorough analysis of the problem of slow BGP convergence upon remote outages, including a survey with 72 operators and measurements on real BGP traces and routers (§2);
- A framework, SWIFT, which enables existing routers to quickly restore connectivity upon such outages (§3);
- Algorithms for quickly inferring disrupted resources from few BGP updates (§4) and enabling fast data-plane rerouting (§5);
- An open-source implementation of SWIFT,1 together with a thorough evaluation (§6) based on real-world BGP traces along with simulations. Among others, we show that SWIFT achieves a prediction accuracy and an encoding efficiency above 90%;
- A case study showing that SWIFT can reduce the convergence time of recent Cisco routers by 98% (§7).

2 THE CASE FOR SWIFT

In this section, we show that slow BGP convergence upon remote outages is practically relevant. We first discuss the causes for slow BGP convergence and its effects on data-plane connectivity in a controlled environment (§2.1). We then present measurements on real BGP traces and feedback from operators: they demonstrate that slow convergence problems occur in the Internet and can lead to significant traffic losses, even for popular destinations (§2.2).

2.1 Slow BGP convergence can cause significant data-plane losses

We describe an example of slow BGP convergence using the network in Fig. 1(a). Each AS i originates a distinct set of prefixes S_i. We focus on the 21k prefixes of S_5, S_7 and S_8, before and after the failure of the link (5, 6). Contrary to what happens in the current Internet (see §2.2.3), we assume that all ASes deploy existing fast-reroute technologies [11, 25, 38, 61]. Those technologies allow each AS to quickly restore connectivity upon a local outage, provided they have a backup path available.

Fig. 1(a) and Fig. 1(b) respectively show pre- and post-failure AS paths. AS 5 knows an alternate path for S_5 (via AS 3) before the failure. However, because of inter-domain policies (e.g., partial

---

1https://github.com/nsg-ethz/swift
transit [60]), it does not know any backup path for $S_6$ and $S_8$ for those prefixes, AS 5 recovers connectivity after the failure via AS 2.

After the failure of $(5, 6)$, AS 5 restores connectivity for $S_7$ almost immediately by rerouting traffic to its alternate path (through AS 3). Since AS 5 does not have backup paths for $S_8$ and $S_9$, a blackhole is created for any flow directed to the corresponding 11k prefixes. In the control plane, the failure causes AS 5 to send 10k path updates to notify that it now uses new paths to reach $S_7$, along with 11k path withdrawals to communicate the unavailability of path $(5, 6, 8)$.

2.1.1 BGP information hiding slows down convergence.

For AS 1 and AS 2, the failure of $(5, 6)$ is a remote outage, which comes with loss of traffic towards $S_5$ and $S_6$. Convergence is inherently slow, since AS 1 and AS 2 only have information about the best paths used by their neighbors and not all the available ones. Upon the failure, AS 1 and AS 2 are indeed forced to wait for the propagation of a large stream of path updates and withdrawals, potentially arriving one prefix at the time. BGP update packing [58] can reduce the number of messages by grouping updates together. However, this mechanism only works if identical BGP attributes are attached to the prefixes to group – which is often not the case due to the widespread use of BGP communities [21]. The absolute number of messages is not the only causes of slow BGP convergence: other reasons include slow table transfer [13], timers [49, 54] and TCP stack implementation [1].

Slow convergence is a fundamental feature of inter-domain routing. Two factors contribute to it. First, routing information must be propagated on a per-prefix basis, because any single AS can apply distinct routing policies, hence use different paths, on a per-prefix basis. Second, routing messages cannot specify network resources that failed, because AS topologies and policies are hidden by the routing system (mainly for scalability and AS-level privacy).

2.1.2 Effect on data-plane connectivity.

To quantify how badly slow control-plane convergence can affect data-plane connectivity, we reproduced the network in Fig. 1 with recent Cisco routers (Cisco Nexus 7000 C7018, running NX-OS v6.2). We then measured the downtime experienced by the AS 1 router upon the failure of $(5, 6)$. In successive experiments, we configured AS 6 to advertise a growing number of prefixes up to 290,000 (roughly half of the current full Internet routing table [5]). As in [26], we injected traffic towards 100 IP addresses randomly selected among prefixes advertised by AS 6, and measured the time taken by AS 1 to retrieve connectivity for all of the probed prefixes. This methodology provided us with a lower bound estimation of the downtime that any prefix in the burst could experience.

Recent routers can lose traffic for dozens of seconds upon remote outages. Table 1 reports the downtime seen by the AS 1 router. Immediately after the link failure, the router starts to drop packets for all monitored IPv4’s. Connectivity is gradually recovered as withdrawals are received from AS 2 and traffic is redirected to AS 3. The evolution of the downtime is roughly linear: for 290k prefixes, the router takes 109 s to fully converge.

<table>
<thead>
<tr>
<th>Withdrawals</th>
<th>Downtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>3.8</td>
</tr>
<tr>
<td>50k</td>
<td>19.0</td>
</tr>
<tr>
<td>100k</td>
<td>37.9</td>
</tr>
<tr>
<td>290k</td>
<td>109.0</td>
</tr>
</tbody>
</table>

Table 1: Data-plane downtime experienced by AS 1 in Fig. 1 as a function of the burst size. Even for relatively small bursts, traffic is lost for tens of seconds.

2.2 Slow BGP convergence in the Internet

We now report evidence of slow BGP convergence in the Internet, along with a discussion on its data-plane impact and on the network operators’ perspectives.

2.2.1 Bursts of withdrawals propagate slowly.

We measured the duration of bursts of BGP withdrawals extracted from 213 RouteViews [51] and RIPE RIS [9] peering sessions during November 2016. We extracted the bursts using a 10 s sliding window: a burst starts (resp. stops) when the number of withdrawals contained in the window is above (resp. below) a given threshold. We choose 1,500 and 9 withdrawals for the start and stop threshold respectively, which correspond to the 99.99th and the 90th percentile of the number of withdrawals received over any 10 s period. Overall, we found a total of 3,335 bursts; 16% of them (525) contained more than 10,000 withdrawals, and 1.5% of them (49) contained more than 100,000 withdrawals. Our measurements expose four major observations.

BGP routers often see bursts of withdrawals. We computed the number of bursts that would be observed by a router maintaining a growing number of peering sessions randomly selected amongst the 213 RouteViews and RIPE RIS peering sessions. Fig. 2(a) shows our results. The line in the box represents the median value, while the whiskers map to the 5th and the 95th percentile. In the median case, a router maintaining 30 peering sessions would see 104 (resp. 33) bursts of at least 5k (resp. 25k) withdrawals over a month. Even if a router maintains a single session, it would likely see a few large bursts each month. Indeed, 62% of the individual BGP sessions we considered saw between 1 and 10 bursts of withdrawals, 24% saw more than 10 bursts. Only 14% of the sessions did not see any. As a comparison, single routers in transit networks routinely maintain tens to hundreds of sessions [27] – even if not all those sessions might carry the same number of prefixes as the ones in our dataset.

Learning the full extent of an outage is slow. While most of the bursts arrived within 10 s, 37% (1,239) of them lasted more than 10 s, and 9.7% (314) lasted more than 30 s (see Fig. 2(b)). This also means that withdrawals within bursts tend to take a long time to be received. In the median case (resp. 75th percentile), BGP takes 13 s (resp. 32 s) to receive a withdrawal.
Large bursts take more time to be learned. Unsurprisingly, large bursts take more time to propagate than smaller ones (see Fig. 2(b)). Overall, we found that 98 bursts took more than 1 min to arrive, with an average size of ≈81k withdrawals.

A significant portion of the withdrawals arrive at the end of the bursts. We took the bursts lasting 10 s or more and divided each of them in three periods of equal duration: the head, the middle and the tail. We found that although most of the withdrawals tend to be in the head of a burst, 50% of the bursts have at least 26% (resp. 10%) of their withdrawals in the middle (resp. in the tail). For 25% of the bursts, at least 32% of the withdrawals are in the tail.

84% of the bursts include withdrawals of prefixes announced by “popular” ASes. We examined the Cisco “Umbrella 1 Million” dataset [6] which lists the top 1 million most popular DNS domains. From there, we extracted the organizations responsible for the top 100 domains: Google, Akamai, Amazon, Apple, Microsoft, Facebook, etc. (15 in total). 84% of the bursts we observed included at least one withdrawal of a prefix announced by these organizations.

2.2.2 Slow BGP convergence lead to significant traffic losses.

While countless studies have shown that BGP convergence can cause long downtime on data-plane connectivity [18, 37, 43, 45, 53, 62, 63, 66], we confirmed the data-plane impact of a few bursts of withdrawals propagated by a national ISP (with more than 50 routers). Specifically, we analyzed the bursts sent by the ISP to its BGP neighbors over a period of three months. Among them, we selected three bursts which included more than 10k withdrawals and which matched with an event logged by the ISP. By checking their logs, the operators identified the root causes of the bursts: two maintenance operations and a peering failure at one of their Internet eXchange Points (IXPs). At least two of these three bursts induced downtime for transit traffic towards up to 68k prefixes, including popular destinations.

2.2.3 Operators care: a 72-operators survey.

To substantiate the effect of slow convergence on operational practices, we performed an anonymous survey among 72 operators coming from a broad variety of networks (ISP’s, CDNs, IXPs, etc.) and providing services to a large customer base: 33% of them connect >1 million users, and 66% connect 10k users or more (see [33] for details). The survey contained 17 questions grouped in three topics: (i) the operators’ experiences with slow convergence; (ii) the induced downtime; and (iii) their opinions on speeding up convergence upon remote outages.

Operators care about slow convergence: 78% of the respondents care about slow BGP convergence and actively aim at decreasing their local convergence time by tweaking the various BGP timers (27 of them) or tuning the underlying TCP parameters (21); 41 respondents use fast-detection mechanisms (BFD) and 21 of them deployed fast-reroute techniques (BGP PIC [25] or MPLS fast-reroute [52]). When considering transit networks (33 respondents), 67% of them rely on fast-detection mechanisms, and 45% on fast-reroute.

Operators often observe slow convergence upon remote outages: Among the 17 respondents who collect statistics about BGP convergence (9 of which are transit ISPs), 52% observe an average BGP convergence time upon remote outages above 30 s. Only 4 of them experience average convergence time below 10 s. In addition, 87% of the respondents observe a worst convergence time above 1 min, and 35% above 5 min.

The vast majority of the operators would be interested in a solution for remote outages like SWIFT: Namely, 95% of our respondents indicated that they would consider adopting a fast-reroute solution to speed up convergence upon remote outages.

3 OVERVIEW

Fig. 3 shows the workflow implemented by a SWIFTED router. We now describe the result of implementing such workflow on the BGP border router of AS 1 in Fig. 1.

Before the outage. The SWIFTED router in AS 1 continuously pre-computes backup nexthops (consistently with BGP routes) to use upon remote outages. This computation is done for each prefix and considering any link on the corresponding AS path. For example, the AS 1 router chooses AS 3 or AS 4 as backup nexthop for rerouting the 20k prefixes advertised by AS 7 and AS 8 upon the failure of link (1, 2). In contrast, it can only use AS 3 as backup to protect against the failure of link (5, 6) for the same set of prefixes, since AS 4 also uses (5, 6) prior to the failure. SWIFT then embeds a data-plane tag into each incoming packet. Each SWIFT tag contains the list of AS links to be traversed by the packet, along with the backup nexthop to use in the case of any link failure.

Upon the outage. After receiving a few BGP withdrawals caused by the failure of (5, 6), the SWIFTED router in AS 1 runs an inference algorithm that quickly identifies a set of possibly disrupted AS links and affected prefixes. The router then redirects the traffic for all the affected prefixes to the pre-computed backup nexthops. To do so, it uses a single forwarding rule matching the data-plane tags installed on the packets. As a result, AS 1 reroutes the affected traffic in less than 2 s (independently from the number of affected prefixes), a small fraction of the time needed by BGP (see Table 1). When rerouting, SWIFT does not propagate any message in BGP. We proved that this is safe provided that the SWIFT inference is sufficiently accurate (§3.3). When BGP has converged, i.e., the burst

[6] Without loss of generality, we assume that a single router in AS 1 maintains all the BGP sessions with AS 2, AS 3 and AS 4.
of withdrawals has been fully received and BGP routes have been installed in the forwarding table, the router removes the forwarding rules installed by SWIFT and falls back to the BGP ones.

In the following, we provide more details about the main components of SWIFT. In §3.1, we overview the inference algorithm (fully described in §4) and how its output is used in a SWIFTED router. In §3.2, we illustrate how SWIFT quickly reroutes data-plane packets on the basis of tags pre-computed by the encoding algorithm detailed in §5. We finally report about SWIFT guarantees in §3.3.

3.1 Inferring outages from few BGP messages

The SWIFT inference algorithm looks for peaks of activity in the incoming stream of BGP messages. Each detected burst triggers an analysis of its root cause. To identify the set of links with the highest probability of being affected by an outage, the algorithm combines the implicit and explicit information carried by BGP messages about active and inactive paths. For example, the failure of (5, 6) in Fig. 1 may cause BGP withdrawals indicating the unavailability of paths (1, 2, 5, 6) and (1, 2, 5, 6, 8) for all the prefixes originated by AS 6 and 8. Receiving these withdrawals makes the algorithm assign a progressively higher failure probability to links in {(1, 2), (2, 5), (5, 6), (6, 8)}. Over time, the algorithm decreases the probability of links (1, 2) and (2, 5), because prefixes originated by ASes 2 and 5 are not affected, and the probability of link (6, 8), because not all the withdrawn paths traverse (6, 8).

SWIFT aims at inferring failures quickly, yet keeping an eye on accuracy. Inference accuracy and speed are conflicting objectives. Indeed, precisely inferring the set of affected AS links might be impossible with few BGP messages, as they might not carry enough information. For instance, SWIFT cannot reduce the set of likely failed links any further than the entire path (1, 2, 5, 6, 8) until it receives other messages than withdrawals for that path. Rerouting based on partial information can unnecessarily shift non-affected traffic, e.g., all the prefixes originated by ASes 2 and 5. In contrast, waiting for BGP messages takes precious time (§2) during which traffic towards actually-affected prefixes can be dropped.

To avoid unnecessary traffic shifts, SWIFT evaluates the likelihood that its inferences are realistic (e.g., using historical data). For instance, SWIFT evaluates the probability that a burst including withdrawals for all the prefixes originated by ASes 6, 7 and 8 happens. If a burst of similar size is unlikely, SWIFT waits for the reception of more messages to confirm its inference. Given that withdrawals for prefixes from AS 7 and 8 will likely be interleaved with path updates for AS 6, this strategy quickly converges to an accurate inference, as we show in §6.

SWIFT uses a conservative approach to translate inferences into predictions of affected prefixes. Remote failures are often partial, that is, an outage can cause traffic loss for a subset of the prefixes traversing the affected link(s). For instance, a subset of the prefixes traversing the failed link (5, 6) in Fig. 1 can remain active because of physical link redundancy between AS 5 and 6, or be rerouted by intermediate ASes (e.g., 5) to a known backup path (like the prefixes originated by AS 7). As BGP messages do not contain enough information to pinpoint the set of prefixes affected by an outage, SWIFT reroutes all the prefixes traversing the inferred links. Doing so minimizes downtime at the potential cost of short-lived path sub-optimality (for a few minutes at most).

SWIFT inference works well in practice. Our experiments on real BGP traces (see §6) show that SWIFT enables to reroute 90% (median) of the affected prefixes after having received a small fraction of the burst, and less than 0.60% of the non-affected prefixes.

3.2 Fast data-plane updates independently of the number of affected destinations

Upon an inference, a SWIFTED router might need to update forwarding rules for thousands of prefixes. In general, routers are slow to perform such large rerouting operations as they update their data-plane rules on a per-prefix basis. Previous studies [24, 64] report median update time per-prefix between 128 and 282 µs. Hence, current routers would take between 2.7 and 5.9 seconds to reroute 21k prefixes (as the router in AS 1 has to do in Fig. 1), and more than 1 minute for the full Internet table (650k prefixes) — even if BGP could converge instantaneously.

SWIFT speeds up data-plane updates by rerouting according to packet tags instead of prefixes. A SWIFTED router relies on a two-stage forwarding table to speed up data-plane updates. The first stage contains rules for tagging traversing packets. SWIFT tags carry two pieces of information: (i) the AS paths along which they are currently forwarded; and (ii) the next-hops to use in the absence (primary next-hops) or presence (backup next-hops) of any AS-link failure. The second stage contains rules for forwarding the packets according to these tags. By matching on portions of the tags, a SWIFTED router can quickly select packets passing through any given AS link(s), and reroute them to a pre-computed next-hop. Since tags are only used within the SWIFTED router, they have local meaning and are not propagated in the Internet (they are removed at the egress of the SWIFTED router).

Using again Fig. 1, we now describe the rules in the forwarding table of the SWIFTED router in AS 1. Fig. 3 shows the tags returned by the SWIFT encoding algorithm. The first stage of the forwarding table contains rules to add tags consistently with the used BGP paths. Since prefixes in AS 8 are forwarded on path (2, 5, 6, 8), it contains the following rule.

```
match(dst_prefix:in AS8) >> set(tag:00011 10011)
```

The first part of the tag identifies the AS path. It maps specific subsets of bits to AS links in a given position of the AS path. The first two bits represent the first link in the AS path, which is link (2, 5). Consistently with Fig. 3, those bits are therefore set to 00. Similarly the second and third bits represent link (5, 6) when it is the second link in the AS path, etc.

The second part of the tag (in green) encodes the primary and backup next-hops. Namely, the first bit identifies the primary next-hop, the second bit indicates the backup next-hop to use if link (1, 2) fails, etc. This part of the tag enables SWIFT to match on traffic that may have to be redirected to potentially different next-hops depending on the link that fails and the destination prefix.

---

3Since the outage affects remote AS links, local fast-rerouting techniques [25] cannot be applied.
Before the failure of (5, 6), the second stage only contains the forwarding rules consistent with BGP. Specifically,

$$\text{match(tag: ***** 1*****)} \gg \text{fwd}(2)$$

Upon the failure of (5, 6), SWIFT adds a single high-priority rule to the second stage – while not modifying at all the first stage.

$$\text{match(tag: *01** 1****)} \gg \text{fwd}(3)$$

The added rule exploits the structure of SWIFT tags to reroute traffic for all the affected 21k prefixes, at once. The regular expression in it matches all the packets such that: (5, 6) appears as the second link in their AS path (i.e., the tag starts with "01**"); and the backup next-hop is 3 (i.e., the tag ends with "****1"). This includes traffic for prefixes in AS 6, 7 and 8. Note that one rule is sufficient in our example, because the SWIFTED router does not use any AS path where (5, 6) appears in other positions before the failure (otherwise, one rule per position would have been needed).

**SWIFT compresses tags efficiently.** Assigning subsets of bits for any AS link and possible position in the AS path does not scale for the Internet AS graph that currently includes >220,000 AS links. SWIFT encoding algorithm squeezes such graph in few bits by leveraging two insights. First, many links in the AS graph are crossed by few prefixes, and their failure does not lead to bursts large enough to even require SWIFT fast-rerouting. SWIFT therefore does not encode those links at all. Second, the AS paths used by a single router at any given time tend to exhibit a limited number of AS links per position. SWIFT therefore only encodes AS links and positions that are present in the used BGP paths.

**SWIFT supports rerouting policies.** When computing backup next-hops, SWIFT complies with rerouting policies specified by the operators. Indeed, rerouting to a safe path may not always be desirable in practice – e.g., because economically disadvantageous. Rerouting policies express the preferences between backup next-hops, or forbid the usage of specific ones—i.e., to mimic business and peering agreements. For example, operators can prevent SWIFT from: (i) using an expensive link with a provider rather than a more convenient one with a customer; (ii) rerouting to a link where free traffic is close to depletion (e.g., according to the 95th percentile rule [50]); or (iii) moving high volumes of traffic to geographically distant regions (e.g., by sending to a remote egress point).

**SWIFT supports both local and remote backup next-hops.** In addition to reroute locally to a directly connected next-hop announcing an alternate route, a SWIFTED router can also fast-reroute to remote next-hops, potentially at the other side of the network, by using tunnels (e.g., IP or MPLS ones). Remote backup next-hops are learned via plain iBGP sessions.

**SWIFT is easy to deploy.** Only a software update is required to deploy SWIFT since recent router platforms readily support a two-stage forwarding table [3]. In §7 we show that SWIFT can also be deployed on any existing router by interposing a SWIFT controller and an SDN switch between the SWIFTED router and its peers. The two-stage forwarding table in that case spans two devices, similarly to an SDX platform [30, 31].

### 3.3 Guarantees and limitations

We prove that SWIFT rerouting strictly improves Internet-wide connectivity, proportionally to the number of SWIFTED routers. This translates into incentives for both partial and long-term Internet-scale deployment (e.g., on all AS border routers).

**Theorem 3.1.** The number of disrupted paths is decreased by every SWIFTED router which is on a path affected by an outage.

**Theorem 3.2.** SWIFT rerouting causes no forwarding loop, irrespective of the set of SWIFTED routers.

Both theorems are based on the following lemma.

**Lemma 3.3.** When any SWIFTED router fast-reroutes, it sends packets over paths with no blackhole and loops.
Proof sketch. Upon a remote outage, any SWIFTED router $r$ reroutes traffic to an AS path that was offered to $r$ by one of its BGP neighbors before the outage (by definition of SWIFT). This path must have been free from blackholes and loops before the outage (by definition of BGP). Also, it contains no failed links—provided that the inference is accurate enough. Hence, the path remains valid and used by all ASes in it, which directly yields the statement. □

As evident from the proof sketch, the lemma and consequently the theorems hold under the following two assumptions (see [33]).

Assumption 1: During an outage, routers only change inter-domain forwarding paths that are affected by the outage. If this assumption is violated, then inter-domain loops can be generated. Let $s$ be a SWIFTED router and $n$ the next-hop to which $s$ fast-reroutes to avoid a certain outage. If $n$ switches path for some fast-rerouted prefixes (e.g., to reflect a policy change uncorrelated with the outage), it may choose the BGP path used by $s$ before the outage (not updated by SWIFT): this would lead to a loop between $n$ and $s$.

Nevertheless, SWIFT can quickly detect and mitigate such a loop: $s$ can monitor whether $n$ stops offering the BGP path to which it has fast-rerouted, and select another backup next-hop. Age), it may choose the BGP path used by $s$ before the outage (not updated by SWIFT). In both cases, packets will be dropped, as it would have happened for the affected prefixes without SWIFT (i.e., using vanilla BGP). However, our evaluation with both real BGP traces and controlled simulations (§6), suggests that very few SWIFT inferences lead to the selection of disrupted backup next-hops.

4 SWIFT INFEERENCE ALGORITHM

We now detail the SWIFT inference algorithm, its basics (§4.1) and how it accounts for real-world factors (§4.2). Because of space constraints, we include the pseudo-code of the algorithm along with the full proof of its correctness (Theorem 4.1) in [33].

4.1 Fast and sound inference

In the following, we consider the stream of messages received on a single BGP session since the algorithm run on a per-session basis (enabling parallelism). We also initially assume that the algorithm aims at inferring an outage produced by a single failed link.

Burst detection. SWIFT monitors the received input stream of BGP messages, looking for significant increases in the frequency of withdrawals. It classifies a set of messages as the beginning of a burst when such frequency (say, number of withdrawals per 10 seconds) in the input stream is higher than the 99.99th percentile recorded in the recent history (e.g., during the previous month).

Failure localization. When detecting a burst, SWIFT infers the corresponding failed link as the one maximizing a metric called Fit Score (FS). Let $t$ be the time at which this inference is done. For any link $l$, the value of FS for $l$ is the weighted geometric mean of the Withdrawal Share (WS) and Path Share (PS):

$$FS(l, t) = (WS(l, t)^{WS} \times PS(l, t)^{PS})^{1/(WS+PS)}$$

WS is the fraction of prefixes forwarded over $l$ that have been withdrawn at $t$ over all the received withdrawals. PS is the fraction of withdrawn prefixes with a path via $l$ at $t$ over the prefixes with a path via $l$. More precisely,

$$WS(l, t) = \frac{W(l, t)}{W(t)} \quad PS(l, t) = \frac{W(l, t)}{W(l, t) + P(l, t)}$$

where $W(l, t)$ is the number of prefixes whose paths include $l$ and have been withdrawn at $t$; $W(t)$ is the total number of withdrawals received as of $t$; $P(l, t)$ is the number of prefixes whose paths still traverse $l$ at $t$. $w_{WS}$ and $w_{PS}$ are the weights we assign to WS and PS. By relying on WS and PS, the fit score aims at quantifying the relative probability that a link is responsible for the received withdrawals while being robust to real-world factors such as BGP noise (§4.2).

Example. Fig. 4 reports the WS and PS values at the end of the burst of withdrawals generated by the failure of (5, 6) in Fig. 1. Link (5, 6) is the only one with both WS and PS equal to 1, since all the AS paths traversing it have been either withdrawn or changed with another path not crossing (5, 6). In contrast, the PS values for links (1, 2) and (2, 5) are smaller than 1 (11k/13k and 11k/12k), because paths for the prefixes of AS 2 and AS 5 have not been modified by the burst. The WS of (6, 8) is smaller than 1 because not all the withdrawals pertain to that link. At the end, (5, 6) is therefore correctly inferred as failed.

SWIFT inference is sound. By soundness, we mean that the inference algorithm is always correct under ideal conditions. The following theorem holds.

Theorem 4.1. If all ASes inject at least one prefix on every adjacent link, SWIFT inference returns a set of links including the failed link if run at the end of the corresponding stream of BGP messages.

Proof sketch. Let $f$ be the failed link and $t$ the time at which all the BGP messages triggered by the failure of $f$ are received.
All prefixes that have been withdrawn were previously forwarded over the $f$, hence $WS(f, t) = 1$. Also, all the prefixes previously forwarded over $f$ have been withdrawn ($PS(f, t) = 1$). This means that the fit score of $f$ has the maximum possible value, hence the SWIFT inference algorithm returns it in the set of inferred links. 

4.2 Robustness to real-world factors
While actual streams of BGP messages do not always match the ideal conditions assumed in Theorem 4.1, SWIFT inferences are good in practice (see §6). We now explain why.

SWIFT makes accurate inferences during the burst. Contrary to the assumptions of Theorem 4.1, SWIFT runs its inference algorithm at the beginning of a burst. Lack of information (i.e., carried by not yet received withdrawals) can therefore affect its accuracy. Being aware of this lack of information, SWIFT uses different weights for $WS$ and $PS$ in the geometric mean calculated in the fit score $FS$ (see §4.1). The key intuition is that early on during the burst, a large number of prefixes are not yet withdrawn and are still using the failed link. As a result, the $PS$ for that link may not be the highest one. The $PS$ for the failed link actually increases when SWIFT runs the inference later in the burst. However, the $WS$ for the failed link will always be greater or equal than the $WS$ of any other link, provided that SWIFT does not receive unrelated withdrawals and that the outage is produced by a single link failure. SWIFT thus performs better when $w_{WS} > w_{PS}$.

By performing a calibration study on real BGP data, we found that SWIFT performed better when $w_{WS}$ was three times higher than $w_{PS}$ (see details in [33]). We therefore use this weight for SWIFT, including in the evaluation (§6).

SWIFT minimizes the risk of inferring a wrong link by being adaptive. As discussed in §3, the accuracy of SWIFT inferences depends on the amount of information in its input.

SWIFT uses the number of withdrawals in an ongoing burst as an estimation of the carried information. It launches a first inference after a fixed number of withdrawals, which we call triggering threshold. If the likelihood of seeing an inferred burst of that size is high enough with respect to historical data, then it returns the inferred link. Otherwise, it waits for another fixed number of withdrawals, and iterates. Using real BGP bursts as baseline (see [33]), we set the default values of the triggering threshold to 2.5k withdrawals. Also, SWIFT returns the inferred link if the number of predicted withdrawals is less than 10k for 2.5k received withdrawals, 20k for 5k received, 50k for 7.5k received, and 100k for 10k received. After having received 20k withdrawals, SWIFT returns the inferred link regardless of the number of predicted prefixes.

SWIFT applies a conservative strategy if failed links cannot be univocally determined. It may happen that SWIFT cannot distinguish precisely which link has failed. For example, in Fig. 4, assuming that the 1k prefixes from AS 6 are updated and not withdrawn. SWIFT cannot distinguish if (5, 6) or (6, 8) failed. Whenever a failed link cannot be univocally determined, SWIFT inference returns all the links with maximum $FS$, i.e., both (5, 6) and (6, 8) in the previous example.

SWIFT quantitative metrics mitigate the effect of BGP noise. Some received BGP messages may be unrelated to the outage causing a burst but due to contingent factors (e.g., misconfiguration, router bugs). They constitute noise that can negatively affect the accuracy of any inference algorithm. In SWIFT, noise can distort $FS$ values. In Fig. 4, for instance, withdrawals for prefixes originated by AS 5 can be received by AS 1 during the depicted burst. This would increase the likelihood that the $FS$ of (2, 5) is higher than the one of (5, 6), especially at the beginning of the burst.

In practice, SWIFT is robust to realistic noise as the level of BGP noise is usually much lower than a burst. Hence, its effect on quantitative metrics like $FS$, $WS$, and $PS$, tends to rapidly drop. This feature distinguishes our inference algorithm from simpler approaches, e.g., based on AS-path intersection, which are much more sensible to single unrelated withdrawals.

SWIFT can infer concurrent link failures. To cover cases like router failures that affect multiple links at the same time, the inference algorithm computes the $FS$ value for sets of links sharing one endpoint. More precisely, the algorithm aggregates greedily links with a common endpoint (from links with the highest $FS$ to those with the lowest one), until the $FS$ for all the aggregated links does not increase anymore. The fit score $FS$ for any set $S$ of links is computed by extending the definition of $WS$ and $PS$ as follows.

$$WS(S, t) = \frac{\sum_{l \in S} W(l, t)}{W(t)} \quad PS(S, t) = \frac{\sum_{l \in S} W(l, t) + P(l, t)}{\sum_{l \in S} W(l, t) + P(l, t)}$$

The set of links (potentially, with a single element) with the highest $FS$ is returned. To ensure safety (see §3.3), for each link inferred, SWIFT must choose a backup route that does not traverse the common endpoint of the links. This prevents SWIFT to reroute a prefix to a backup next-hop that uses another inferred link (because all the inferred links have a common endpoint). By choosing backup paths bypassing a superset of the inferred links, SWIFT also ensures safety in case the inference algorithm correctly localizes the ASes involved in the outage instead of the precise links.

5 SWIFT ENCODING ALGORITHM
In this section, we describe how SWIFT tags are computed. Recall that these tags are embedded onto the incoming packets in the first stage of the forwarding table and are split in two parts: one which encodes the AS links used by the packet, and another which encodes the next-hops to reroute to should any of these links fail. Thanks to these embedded tags, a SWIFTED router can reroute traffic efficiently upon an inference, independently on the number of prefixes impacted.

In section 7, and similarly to [30, 31], we show how SWIFT can leverage the destination MAC to tag incoming traffic. The destination MAC is indeed a good “tag carrier” as it provides a significant number of bits (48), and can easily be removed in the second stage of the forwarding table by rewriting it to the MAC address of the actual next-hop, as any IP router would do.

---

[3] This is enough to ensure safety. However, SWIFT computes the backup next-hops in advance, i.e., before the failure (see §3.2). As SWIFT does not know which endpoint of a link will be the common endpoint, it chooses backup paths (for the prefixes traversing this link) avoiding both endpoints of the link.
Encoding AS links. The first part of the tag (right side of Fig. 5) encodes the AS path along which each packet will flow. For each prefix, we consider the AS path associated with the best route for it, and we store the position of ASes in that path. Namely, we define \( m \) sets, with \( m \) being the length of the longest AS path, and we call the \( i \)-th set position \( i \). For any AS path \( (u_0 u_1 \ldots u_k) \), with \( k \leq m \), we then add the AS identifier of \( u_i \) to position \( i \), for every \( i = 1, \ldots, k \). Note that the first hop in any AS path is already represented as primary next-hop (see Part 2 of Fig. 5). Hence, we do not model position 1, and we have a different AS-path encoding for every SWIFT’s neighbor. At the end of this process, AS paths can be encoded by selecting specific AS identifiers for every position.

Encoding all used AS paths may not be possible. Not only can thousands of distinct ASes be seen for each position, but also the AS paths may be very long (>10 hops). Fortunately, two observations enable SWIFT to considerably reduce the required number of bits. First, from the perspective of one router, many AS links carry few prefixes. A failure of these links will therefore produce small bursts (if any), which allows for per-prefix update. Thus, we ignore any link that carries less than 1,500 prefixes in our SWIFT encoding. Second, links that are far away from the SWIFT node are less likely to produce bursts of withdrawals than closer ones. Indeed, for distant links, it is likely that intermediate nodes know a backup path. Our measurements (§6) confirm this. Consequently, we only encode the first few hops of the AS paths (up to position 5).

For the remaining AS links, SWIFT encodes first the links with the highest number of prefixes traversing them. To do that, SWIFT uses an adaptive number of bits for each AS position: each position is implemented by a different bit group, whose length depends on the number of ASes in this position. For each position \( P \), we map all the ASes in \( P \) to a specific value (the AS identifier) of the corresponding bit group. Hence, the size of this group is equal to the number of bits needed to represent all the values in \( P \).

Encoding backup next-hops. The second part of the tag (left side of Fig. 5) identifies the primary next-hop as well as backup next-hops for each encoded AS link. For each prefix \( p \), the primary next-hop is directly extracted as the first hop in the AS path for \( p \). For instance, the primary next-hop for prefix \( p_1 \) in Fig. 1(a) is 2. Backup next-hops are explicitly represented to both reflect rerouting policies and prevent rerouting to disrupted backup paths. Consider again \( p_1 \). The primary path is \( 2, 5, 6 \). To protect against a failure of the first AS link \( (2, 5) \), we can select AS 3 or 4, since neither of the two uses \((2, 5)\) to reach \( p_1 \). In contrast, for \((5, 6)\), only AS 3 can be used as a backup next-hop, since the AS paths received from AS 4 also uses \((5, 6)\).

Partitioning bits across the two parts of the tag. A fundamental tradeoff exists between the amount of paths and the number of backup next-hops that any SWIFT router can encode. On the one hand, allocating more bits to represent AS links (first part of the tag) allows a SWIFT router to cover more remote failures. On the other hand, allocating more bits to represent (backup) next-hops (second part of the tag) allows a SWIFT router to reroute traffic to a higher number of backup paths.

In §6.4, we show that allocating 18 bits to AS paths encoding is sufficient to reroute more than 98% of the prefixes. Assuming 48-bits tags (i.e., using the destination MAC), 30 bits are left to encode backup next-hops. If we configure SWIFT to support remote failures up to depth 4, the bits allocated for the backup next-hops needs to be divided by 5 (1 primary + 4 backup next-hops). As a result, \( 30/5 = 6 \) bits are reserved for each depth, which translates into \( 2^6 = 64 \) possible next-hops. If one wants to consider remote failures only up to depth 3, then the number of next-hops is \( 2^5 = 128 \) and two more bits can be allocated to the AS links encoding. Operators can fine-tune such decision, e.g., based on the (expected) number of backup next-hops reachable by each SWIFT router.

6 EVALUATION

We now evaluate our Python-based implementation (= 3,000 lines of code) of the SWIFT inference algorithm (§4) and the encoding scheme (§5). We first describe our datasets (§6.1). We then evaluate the accuracy of the inference algorithm, both in terms of failure localization (§6.2) and withdrawals prediction (§6.3). We also evaluate the efficiency of SWIFT data-plane encoding (§6.4). Finally, we show that the combination of the inference algorithm and the encoding scheme leads to much faster convergence than BGP (§6.5).

6.1 Datasets

We evaluate SWIFT using two sources of bursts of BGP withdrawals.

Bursts from real BGP data, without outage ground truth. To evaluate how SWIFT would work in the wild, we use sets of actual bursts extracted from the same dataset used in §2. It consists of BGP messages dumped by 10 RouteViews [51] and 5 RIPE RIS [9] collectors during the full month of November 2016. These collectors received BGP messages from 213 peers. Our evaluation is based on 1,802 bursts with more than 1,500 withdrawals. Amongst them, 942 (resp. 339) have more than 2,500 (resp. 15,000) withdrawals.

We found 5 routers peering with these collectors that exhibit a flapping behavior, with an anomalous large number of bursts of similar pattern; when including them, we obtain a minimal change in overall results (≈2%), but since SWIFT performs uniformly on similar bursts, their large number (=500 bursts) causes a significant skew in the population of bursts. We therefore omit these peers from our analysis.
6.2 Failure localization accuracy

In the following, we evaluate the accuracy of the SWIFT inference algorithm on both datasets.

6.2.1 Validation on real BGP data.

Since real BGP traces do not provide the ground truth on burst root causes, we estimate the accuracy of the inference algorithm indirectly: we evaluate the match between the prefixes withdrawn in the entire burst $W$ and the prefixes $W'$ whose path traversed the links inferred by SWIFT as failed. This can be formalized as a binary classification problem, in which the true and false positives are the prefixes in $W' \cap W$ and $W' - W$, respectively. We therefore evaluate the accuracy of SWIFT inference in terms of True Positive Rate (TPR) and False Positive Rate (FPR).

Fig. 6 shows the TPR and FPR on a per-burst basis. It is divided into quadrants. The top left quadrant corresponds to very good inferences, i.e., for each burst, the links that SWIFT infers as failed are traversed by most of the withdrawn prefixes (high TPR) and few of the non-affected prefixes (low FPR). The top right quadrant contains inferences that overestimate the extent of a failure (high TPR and FPR): rerouting upon such inferences is still beneficial as the TPR is high (i.e., connectivity is restored for many prefixes actually disrupted). The bottom left quadrant corresponds to inferences that underestimate the extent of a burst. Finally, the bottom right quadrant includes bad inferences (with low TPR and high FPR).

We evaluate two scenarios for SWIFT. In the first one (Fig. 6(a)), the inference algorithm runs only once, after 2.5k withdrawals—as it would do without a history model (e.g., after the first installation on a router). In the second scenario (Fig. 6(b)), the inference algorithm runs every 2.5k withdrawals while following the simple historical model we described in §4.2. When considering history, SWIFT waits for more withdrawals to arrive before rerouting large numbers of prefixes early on in the burst.

We now describe the results obtained by SWIFT inference algorithm when run on the bursts generated in C-BGP (see §6.1).

Under ideal conditions, SWIFT inference is always correct. We ran our inference algorithm at the end of each burst and found that the inference is always correct, consistently with Theorem 4.1.

SWIFT inference is accurate enough to ensure safety, even early on during the bursts. When we ran the inference algorithm after only 200 withdrawals (1% of the total number of prefixes advertised, see §6.1), SWIFT identified a superset of the failed link for 9% of the bursts. For the remaining 91%, it returned a set of links adjacent to the failed one. Nevertheless, for all the 2,183 bursts but one, SWIFT selected a backup path that bypasses the actual failed link. This is because SWIFT chooses a backup route that does not traverse the common endpoint of the inferred links (see §4.2).

**Figure 6:** Despite having little information, SWIFT inference is accurate. The vast majority of prefixes are correctly inferred as failed (top half quadrants). While some affected prefixes are missed (bottom left), no prediction is significantly inaccurate (bottom right).
with CP and FP, the total numbers of prefixes correctly predicted and withdrawn.

<table>
<thead>
<tr>
<th>Burst size between 2.5k and 15k</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPR</td>
</tr>
<tr>
<td>FPR</td>
</tr>
<tr>
<td>CP</td>
</tr>
<tr>
<td>FP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Burst size greater than 15k</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPR</td>
</tr>
<tr>
<td>FPR</td>
</tr>
<tr>
<td>CP</td>
</tr>
<tr>
<td>FP</td>
</tr>
</tbody>
</table>

Table 2: Inference algorithm with history model: performance of the prediction of future withdrawals.

**SWIFT inference is robust to noise.** We simulated BGP noise by adding, in each burst, 1,000 withdrawals of prefixes that are not affected by the failure. This number is much greater than what we observe in real BGP data, both in absolute terms (9 withdrawals only in the 90th percentile, see §2.2) and as a percentage (since we only advertise 20k prefixes in C-BGP, whereas there are more than 600k prefixes advertised in the real world [5]). We triggered the inference at the end of the burst, SWIFT identified the failed link for 91% of the bursts (991), a superset for 9% bursts (188), a set of links adjacent to the failed one for 1 burst and did a wrong inference for 3 bursts. When we triggered the inference after 200 withdrawals, SWIFT still selected backup paths that bypass the actual failed link for all the bursts but one. SWIFT identified a superset of the failed link for 12% of the bursts, while for the remaining 88%, it returned a set of links adjacent to the failed one.

### 6.3 Withdrawals prediction accuracy

In the previous section (§6.2), we showed that SWIFT inference algorithm is indeed able to identify the failed link, even with limited information. In this section, we evaluate the ability of SWIFT to predict withdrawals, we also give the absolute number of prefixes fast rerouted upon such inference, enabling us to quantify the benefit of SWIFT, as well as the possible under/overshooting induced.

Differently from the previous section, in order to evaluate specifically the prediction, we consider as “positives” only the prefixes withdrawn after the inference was made. This change affects the definition of TP (and TPR) but leaves FP (FPR) unaltered. Since we already used TPR in §6.2, we denote with CPR (for Correctly Predicted Rate) the true positive rate of the prediction. We also denote with CP and FP, the total numbers of prefixes correctly predicted or not, respectively.

6.3.1 Validation on real BGP data.

Table 2 shows results obtained by running the SWIFT inference algorithm with the history model. Results for small (≤15k) and large (>15k) bursts are shown separately.

**SWIFT correctly fast-reroutes a large number of affected prefixes.** For half (resp. 80%) of the small bursts, SWIFT correctly predicts at least 89.5% (resp. 48.9%) of the future prefix withdrawals. For half (resp. 80%) of the large bursts, SWIFT correctly predicts at least 93% (resp. 39.3%) of the future prefix withdrawals. In terms of absolute numbers, we see that SWIFT correctly fast-reroutes a significant amount of prefixes, especially for larger (>15k) bursts, where the number of prefixes predicted is in the order of tens of thousands for 60% of the bursts and in the order of hundreds of thousands for more than 10%.

**SWIFT only reroutes a small number of non-affected prefixes.** Both for small and large bursts, the fraction of fast-rerouted prefixes that were not affected by the failure is small in most of the cases. In few cases (e.g., 90-th percentile of the large bursts) however, the algorithm significantly overestimates the number of prefixes to be rerouted (FP). This is because we deliberately designed and tuned the algorithm to not minimize incorrectly rerouted prefixes in order to avoid missing prefixes that should be rerouted. Incorrectly rerouted prefixes are indeed forwarded to a backup path which is sub-optimal but not disrupted, just for the few minutes needed for BGP to reconverge. Consistently, we note that less aggressive weights do reduce the FPR (see [33]).

6.3.2 Validation through simulation.

We now evaluate the accuracy of the prefixes prediction on the bursts generated by C-BGP.

**SWIFT accurately predicts prefix withdrawals, even when considering noise.** When inferring the affected prefixes after only 200 withdrawals, the FPR is equal to 0% for 98% of the bursts. The highest FPR observed is only 13%. In the median case (resp. 25th percentile), the CPR is equal to 88% (resp. 84%). The lowest CPR observed is 37%.

To consider the impact of BGP noise on these numbers, we added, to each burst, 1,000 withdrawals unrelated to the failure (as in §6.2.2). We found that, for 53% of the bursts, the FPR is still 0%. The FPR is greater than 9% for only 1% of the bursts. In the median case (resp. 25th percentile), the CPR is 53% (resp. 50%). The CPR is far from 100% because the withdrawals unrelated to the failure count as positives. In practice, we observe that the CPR is less affected by BGP noise, as the level of noise is usually much lower (see §2.2).

### 6.4 Encoding effectiveness

We now experimentally evaluate SWIFT encoding scheme (§5) by quantifying how many prefixes can effectively be rerouted in the data-plane by matching on the pre-provisioned tags. For each burst, we define the **encoding performance**, as the fraction of predicted prefixes that can be rerouted by the encoding scheme. The performance depends on the number of bits allocated to the AS path part of the tag (see §5). For this part of the evaluation, we rely on the inference algorithm with the history model and consider the bursts obtained from the real BGP data.
Allocating 18 bits to the AS-path part of the tag enables to reroute 98.7% of the predicted prefixes. Fig. 7 shows the encoding performance (over all bursts) as a function of the number of bits reserved for the AS-path part of the tag. Each box shows the inter-quartile range of the encoding performance: the line in the box depicts the median value; the dot depicts the mean; and the whiskers show the 5th and 95th percentiles. As the number of bits allocated to the AS paths encoded increases, so does the encoding performance. We see that 18 bits are already sufficient to reroute 98.7% of the predicted prefixes in the median case (73.9% in average). These results illustrate that the compression done by the encoding algorithm is efficient and manages to encode the vast majority of the relevant AS links. In addition, Fig. 7 shows that for the large bursts of at least 10k withdrawals, the encoding performance is better (84.0% on average with 18 bits). This is explained by the design of our encoding algorithm, which encodes with highest priority the AS links with the largest number of prefixes traversing them (and which may cause large bursts in case of a failure).

Assuming a tag of 48 bits (e.g., using the destination MAC), the remaining 30 bits can be used to encode the backup next-hops. If SWIFT encodes up to depth 4 (i.e., position 5 in the AS path), 64 different next-hops can therefore be used. This suggests that SWIFT encoding can work well even if the SWIFTED device is connected to a large number of external neighbors, like in IXPs [47]. The number of backup next-hops can even be increased by reducing the number of AS hops encoded (e.g., up to depth 3 instead of 4).

### 6.5 Rerouting speed

In this section, we show that the combination of the SWIFT inference algorithm and the encoding scheme enables fast convergence in practice (within 2 s) by quantifying: (i) the learning time required for a prediction; and (ii) the number of rules updates to perform in the data plane. Our results are computed on the bursts in the real BGP data.

SWIFT learns enough information to converge within 2 seconds (median). Compared to vanilla BGP, SWIFT converges much faster than a BGP router working at the per-prefix level. Fig. 8 shows the CDF of the time elapsed between the beginning of the burst and the actual time at which every withdrawal in the burst is learned. For BGP, the learning time corresponds to the withdrawal timestamp. For SWIFT, it corresponds to the prediction time if the withdrawal is predicted, otherwise the withdrawal timestamp. The plot highlights that, in the median case, SWIFT learns a withdrawal within 2 s, while BGP needs 13 s. We can observe a shift at 41 s in the SWIFT curve. After investigation, we found that this is due to a very large burst of 570k withdrawals which took a total of 105 s to arrive. The first 20k withdrawals (needed for SWIFT to launch the prediction) took 41 s to arrive. Observe that, even in such a case, SWIFT was still able to shave off more than 1 min of potential downtime.

**SWIFT requires few data-plane updates to reroute all the predicted prefixes.** The number of data-plane updates required to reroute all the predicted prefixes depends on the number of failed AS links reported by the inference algorithm. When executing the inference algorithm after 2.5k withdrawals, in 29% of the cases, the number of links predicted is 1 and the median number (resp. 90th percentile) is 4 (resp. 29). For each reported link, one data-plane update is required for each backup next-hop (§5). As a result, in the median case (resp. 90-th percentile) and with 16 backup next-hops, 64 (resp. 464) data-plane updates are required. Considering a median update time per-prefix between 128 and 282 µs [24, 64], SWIFT can update all the forwarding entries within 130 ms.

### 7 CASE STUDY

In this section, we showcase the benefits of SWIFT by boosting the convergence time of a recent Cisco router. As mentioned in §3.2, SWIFT can be implemented directly on existing routers via a simple software update, since the only hardware requirement, a two-stage forwarding table, is readily available in recent platforms [3] (we confirmed this implementation through discussion with a major router vendor). Yet, to evaluate SWIFT without waiting for vendors to implement it, we developed an alternative deployment scheme.

**How to SWIFT any existing router.** In our alternative deployment scheme, we interpose a SWIFT controller and an SDN switch between the SWIFTED router and its peers, respectively at the control- and data-plane level (as in Fig. 9(b)). The setup is akin to the SDX platform [30, 31]. It enables to deploy SWIFT on any router that supports BGP and ARP, that is, virtually any router.

Upon reception of the BGP updates coming from the peers of the SWIFTED router, the controller assigns 48-bit tags according to...
the SWIFT encoding scheme (see §5). The controller programs the SWIFTed router to embed the data-plane tags in the destination MAC field in the header of incoming packets, using the same technique as in a SDX [31] (i.e., with BGP and ARP). It also programs the SDN switch to route the traffic based on the tags, and rewrite the destination MAC address with the one of the actual next-hop. The two-stage forwarding table used by SWIFT then spans two devices: the SWIFTed router (first stage) and the SDN switch (second stage).

Upon the detection of a burst coming from a peer, the SWIFT controller runs the inference algorithm (§4), and provisions data-plane rules to the SDN switch for rerouting the traffic. Our SWIFT controller uses ExaBGP [7] to maintain BGP sessions.

Methodology. We reproduced the topology in Fig. 1(a) with a recent router (Cisco Nexus 7k C7018, running NX-OS v6.2) acting as AS 1, which we connected to its peers via a laptop running a (software-based) OpenFlow switch (OpenSwitch 2.1.3). We configured AS 6 to announce 290k prefixes. Then, we failed the link (5, 6), and we measured the downtime using the same technique as in §2 (sending traffic to 100 randomly selected IP addresses).

A 98% speed-up. Fig. 9(a) reports the downtime observed by the SWIFTed and non-SWIFTed Cisco router. While the vanilla Cisco router takes 109 s to converge, the SWIFTed Cisco router systematically converges within 2 s—a 98% speed-up.

8 RELATED WORK

Root Cause Analysis (RCA). Many prior works aim at identifying the root cause of failures, be it in the Internet [14, 15, 19, 23, 36, 39, 40, 68–70], or within a network [20, 41, 59, 67]. SWIFT inference algorithm differs from previous works both in objectives and scope. To enable fast rerouting, SWIFT inference should be extremely quick (in seconds or sub-seconds), while previous works typically focus on a much longer timescale (minutes). Moreover, SWIFT deals with a specific type of failures, those generating large bursts of BGP withdrawals, and only rely on the BGP messages reaching a single vantage point (the SWIFTed router). In contrast, previous RCA efforts typically use active measurements and multiple vantage points. They also focus on pinpointing different problems such as per-prefix path changes [36] or failures on the reverse path [39].

Another important difference is that SWIFT actually uses its fast RCA core to repair Internet connectivity problems (almost in real time). Doing so goes beyond previous contributions, like [34], which only show how to detect (not repair) path problems out of passive packet-level traces collected from a single vantage point.

BGP convergence. Slow BGP convergence is a well-known problem [17, 22, 29, 44, 45, 48]. Most prior work aimed at reducing BGP convergence time within a single domain, for instance, upon planned maintenance or internal link failures. For example, LOUP [32] improved internal BGP convergence by ordering external route updates to avoid transient loops. SWIFT complements and generalizes these approaches by speeding-up local rerouting upon remote failures. SWIFT goals are similar to R-BGP [44] which enables faster failover in inter-domain routing by pre-computing and propagating few disjoint failover paths. Unlike SWIFT though, R-BGP is not compatible with existing routers: it may also require many paths to be propagated Internet-wide and stored in routers.

BGP burstiness. Several works [15, 23, 46, 49] focused on bursts of BGP messages with the goal of studying per-prefix instabilities and dynamics. They define an update burst as a sequence of messages pertaining to a single prefix and observed within a given timeout. Our goal with SWIFT is different, as we focus on events generating concurrent withdrawals related to distinct prefixes.

Fast data-plane updates. Several techniques can speed up forwarding rule modification upon local failures. For example, MPLS fast reroute [52], IP fast reroute [11, 61] and PIC [25] can react in sub-second to local link failures by pre-provisioning backup entries and selectively activate them at runtime. SDN approaches, like Fat-Tire [57], support the same use case in OpenFlow. None of these works can fast reroute upon remote failures—as SWIFT does.

9 CONCLUSION

We presented SWIFT, the first fast-reroute framework for remote outages. SWIFT is based on two key contributions: (i) a fast and accurate inference algorithm; and (ii) a novel encoding scheme. We performed a thorough evaluation of SWIFT using a fully functional implementation and real BGP data. Our results indicate that SWIFT is efficient in practice: it achieves a prediction accuracy and an encoding efficiency both above 90%, and can boost the convergence performance of a Cisco router by up to 98%.

Acknowledgements

We are grateful to the SIGCOMM reviewers and our shepherd, Kun Tan, for their insightful comments and suggestions. We thank Derk Valenkamp and Paul Stark from the ICT-Networks team at ETH Zürich for providing us with access (and support) to their networking gear. Thanks also to Alexander Gall, Chris Welti and Simon Leinen for giving us access to BGP data and logs. Finally, we thank Jennifer Rexford and Olivier Bonaventure for their useful comments on an earlier version of the paper. This work was partially supported by the US Department of Homeland Security Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD), via contract HHSP 233201600012C.