SWIFT: Predictive Fast Reroute

Technical Report

Thomas Holterbach[*][†], Stefano Vissicchio[‡], Alberto Dainotti[†], Laurent Vanbever[*]
[*]ETH Zürich, [†]CAIDA, UC San Diego, [‡]University College London

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 [54].

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, 34, 64], 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 routinely 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, 36, 42, 44, 52, 61, 62, 65]), 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 testbed 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, 35, 38, 39, 67–69]), 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].

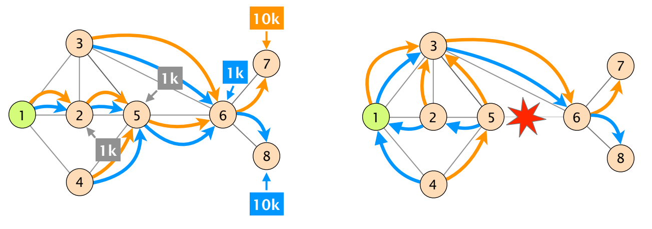
Whenever a SWIFTED router fast-reroutes upon an outage, it guarantees connectivity to all the traffic sources passing through it. Hence, deploying SWIFT in a few central ASes would benefit the entire Internet, since these ASes would also protect their (non-SWIFTED) customers. The same applies within 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%;

[1]https://github.com/nsg-ethz/swift



(a) Initial forwarding paths (b) Post-failure paths

Figure 1: Example of slow convergence upon a remote outage: routing policies and absence of information about physical connectivity force AS 1 to wait for 11k BGP withdrawals, one per prefix owned by AS 6 or AS 8.

- 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_6, 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, 37, 60]. 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_7 (via AS 3) before the failure. However, because of inter-domain policies (*e.g.,* partial transit [59]), 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_6 and S_8, 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_6 and S_8. 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.

Withdrawals	Downtime (sec)
10k	3.8
50k	19.0
100k	37.9
290k	109.0

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.

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 [57] 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 [48, 53] 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 IPs. 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.

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 [50] 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.

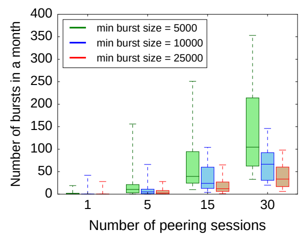
BGP routers often see bursts of withdrawals. We computed the number of bursts that would be observed by a router maintaining a growing number of peering sessions randomly selected amongst the 213 RouteViews and RIPE RIS peering sessions. Fig. 2(a) shows our results. The line in the box represents the 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.

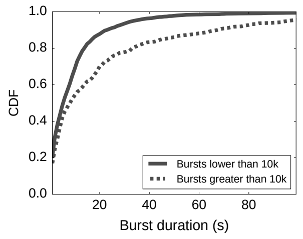
Large bursts take more time to be learned. Unsurprisingly, large bursts take more time to propagate than smaller ones (see Fig. 2(b)). Overall, we found that 98 bursts took more than 1 min to arrive, with an average size of ≈81k withdrawals.

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.



(a) Bursts of withdrawals are frequent and involve a high number of prefixes, including popular ones.



(b) Bursts take a long time to propagate; longer bursts even more.

Figure 2: Size and duration of bursts captured from 213 BGP vantage points in November 2016.

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, 36, 42, 44, 52, 61, 62, 65], 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 problem of slow convergence on operational practices, we performed an anonymous survey among 72 operators. The survey contained 17 questions grouped in three main topics: *i)* the operators' perception on slow BGP convergence (do they care?); *ii)* the duration of BGP-induced downtime they usually experience; and *iii)* their views on speeding-up convergence upon remote outages (would they do it?).

Breakdown of respondents: Our respondents come from a wide variety of networks providing one or more services to a large customer base. The majority of the respondents (67%) provides Internet connectivity to end-users (wired or mobile). 46% of them provides transit services. 19% of them work for Content Distribution Networks (CDN), while 15% of them work for an Internet Exchange Points (IXP). In terms of customer base, 33% of our respondents connect 1 million or more users to the Internet, 48% connect 100,000 users or more, while 66% connect at least 10k users. 76% of our respondents work with full Internet routing tables, meaning that their routers carry more than 650k prefixes [5] in their forwarding tables.

Operators care about slow convergence: 78% of the respondents care about slow BGP convergence. The remaining 22% do not care because they: receive a single default route from their provider (3 of them); do not have stringent Service Level Agreement to meet (6 of them); or because they have never experience a slow convergence before (4 of them).

76% of the respondents actively aim at decreasing their local convergence time by: tweaking the different BGP timers used by the routers (27 of them) or tuning the BGP transport parameters (21 of them); 40 respondents use fast-detection detection mechanisms (BFD) and 21 of them deployed fast-reroute techniques (BGP PIC or MPLS fast-reroute). When considering only transit networks (33 respondents), 67% of them rely on fast-detection detection mechanisms, and 45% of them, on fast-reroute techniques.

Few operators monitor BGP-induced downtime (it is hard), but those who do routinely experience slow convergence upon remote outage: The majority (76%) of the operators do not collect statistics about BGP convergence as it is hard in practice (we provide such measurements in §2.2). Among the 17 operators who do (9 of which are transit ISPs), the majority (52%) observe *average* BGP convergence time upon remote outage above 30 seconds. Only 4 of them experience average convergence time below 10 seconds. Regarding *worst-case* convergence time upon remote outages, 87% (resp. 35%) of the respondents observe convergence time above 1 (resp. 5) minutes.

The vast majority of the operator would adopt a solution solving remote outages like SWIFT: 67 of our respondents (95%) indicated that they would consider adopting a fast-reroute solution to speed-up remote outage such as

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[2] of AS 1 in Fig. 1.

Before the outage. The SWIFTED router in AS 1 continuously pre-computes backup next-hops (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 next-hop 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 next-hop 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 next-hops. 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.

[2]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.

We proved that this is safe provided that the SWIFT inference is sufficiently accurate (§3.3). When BGP has converged, *i.e.,* the burst 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.[3] Previous studies [24, 63] 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:00111 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.

[3] Since the outage affects remote AS links, local fast-rerouting techniques [25] cannot be applied.

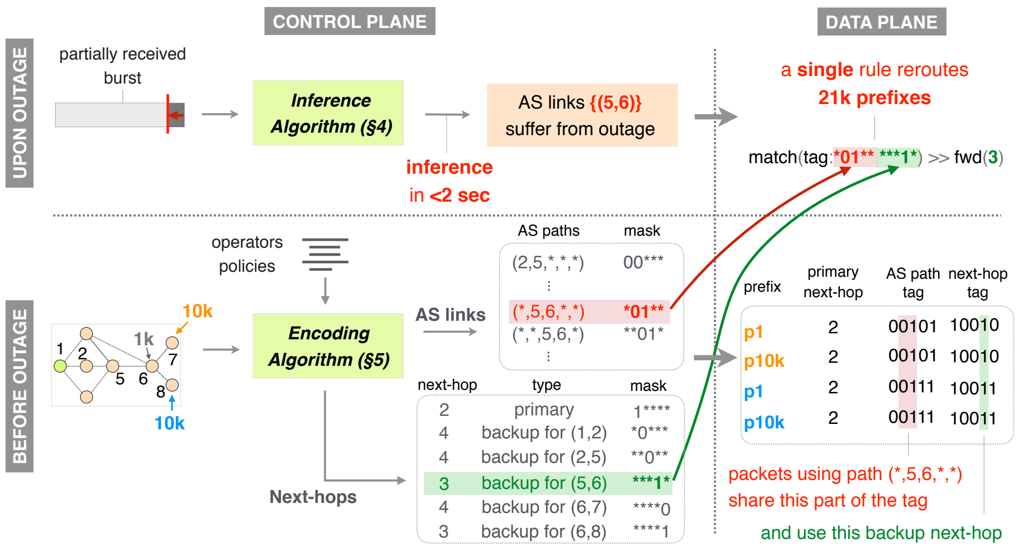


Figure 3: SWIFT workflow.

Before the failure of (5, 6), the second stage only contains the forwarding rules consistent with BGP. Specifically,

```
match(tag:***** 1****) >> 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.

```
match(tag:*01** ***1*) >> 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 [49]); 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 Appendix B).

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.

Assumption 2: SWIFT inferences enable the SWIFTED routers to avoid paths affected by an outage. The SWIFT inference algorithm implements a conservative approach for inferring links and selecting backup paths. Still, we cannot guarantee the validity of such assumption, since SWIFT inferences are based on the partial and potentially noisy information provided by BGP (and withdrawals that reach different ASes at different times). Inferences that cause SWIFT not to rule out all paths affected by an outage might induce packet loss: in these cases, a SWIFTED router could reroute traffic to a disrupted backup, and multiple SWIFTED routers could create an inter-domain loop (if the selected backup next-hop actually uses exactly one of the disrupted paths missed by the inference). 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.

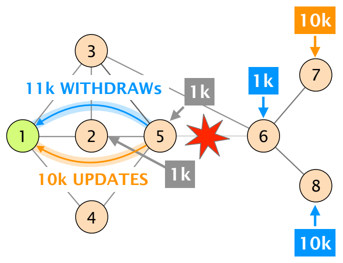
4 SWIFT INFERENCE ALGORITHM

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 Appendix A and B.

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).



link	WS	PS
(1,2)	11/11	11/13
(2,5)	11/11	11/12
(5,6)	11/11	11/11
(6,7)	0/10	0/10
(6,8)	10/11	10/10

Figure 4: WS and PS metrics at the end of the burst of withdrawals caused by the failure of (5,6).

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):

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

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 at t. More precisely,

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

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 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 Appendix C). 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 Appendix C), 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.

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

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.[4] 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.

[4]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.

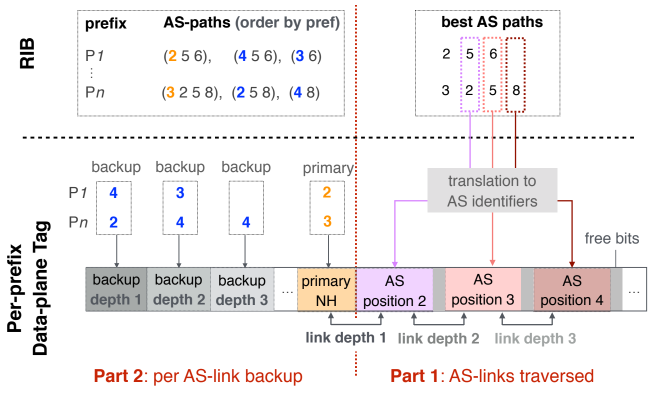


Figure 5: A SWIFTED router embeds a tag into incoming packets. The tag encodes the links traversed by the packet (Part 1) along with backup next-hops for each of the encoded links (Part 2).

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 $(u_0\ u_1 \dots u_k)$, with $k \leq m$, we then add the AS identifier of u_i to position i, for every $i = 1, \dots, k$. 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 SWIFTED 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 $p1$ 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 $p1$. 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 $p1$. 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 SWIFTED 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 SWIFTED 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 $2^7 = 128$ 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 SWIFTED router.

6 EVALUATION

We now evaluate our Python-based implementation ($\approx$ 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 [50] and 5 RIPE RIS [9] collectors during the full month of November 2016. These collectors received BGP messages from 213 peers.[5] 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.

Bursts from simulations, with outage ground truth. To validate the accuracy and the robustness of our inference algorithm,

[5]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 ($\approx$2%), but since SWIFT performs uniformly on similar bursts, their large number ($\approx$500 bursts) causes a significant skew in the population of bursts. We therefore omit these peers from our analysis.

we use bursts extracted from control-plane simulations conducted with C-BGP [55]. We created a topology composed of 1,000 ASes using the Hyperbolic Graph Generator [10]. We set the average node degree to 8.4, which is the value observed in the CAIDA AS-level topology [16] in October 2016, and use as degree distribution a power law with exponent 2.1 [41]. We defined the AS relationships as follows. The three ASes with highest degree are Tier1 ASes and are fully-meshed. ASes directly connected to a Tier1 are Tier2s. ASes directly connected to a Tier2 but not to a Tier1 are Tier3s, etc. Two connected ASes have a peer-to-peer relationship if they are on the same level, otherwise they have a customer-provider relationship. We configured each AS to originate 20 prefixes, for a total of 20k prefixes. Using C-BGP, we simulated random link failures, and recorded the BGP messages seen on each BGP session in the network. We collected a total of 2,183 bursts of at least 1k withdrawals. The median (resp. max) size of the bursts is 2,184 (resp. 19,215) withdrawals.

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.*

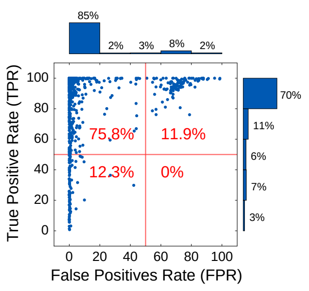
Since real BGP traces do not provide the ground truth on burst root causes, we estimate the accuracy of the inference algorithm indirectly: we evaluate the match between the prefixes withdrawn in the entire burst W and the prefixes $W^{'}$ whose path traversed the links inferred by SWIFT as failed. This 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).[6]

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).

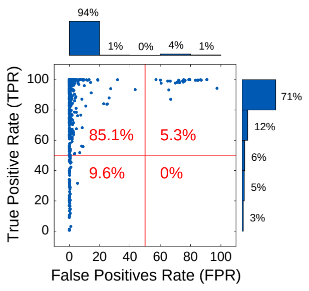
We evaluate two scenarios for SWIFT. In the first one (Fig. 6(a)), the inference algorithm runs only once, after 2.5k withdrawals—as it would do without a history model (*e.g.,* after the first installation on a router). In the second scenario (Fig. 6(b)), the inference algorithm runs every 2.5k withdrawals 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.

SWIFT makes accurate inferences in the majority of the cases, and never makes *bad* inferences. Even when using only 2.5k withdrawals (Fig. 6(a)), SWIFT makes accurate inferences in the

[6]$TPR = TP/(TP + FN)$, $FPR = FP/(FP + TN)$; The negatives are all the prefixes announced in the session before the burst starts and not withdrawn during the burst.



(a) Without history



(b) With history

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).

vast majority of the cases: TPR is more than 60% for more than 81% of the bursts. However, it also overestimates the extent of the burst (FPR is higher than 50%) for about 12% of the bursts. SWIFT inference algorithm performs sensibly better when relying on history (Fig. 6(b)). Better performance comes at the price of missing some bursts because of the extra delay. Specifically, it missed a total of 256 bursts (53% of them smaller than 5k) compared to the history-less version. Despite this, the history-based version of the inference algorithm still completes the inference at the lowest threshold (2.5k) for the majority of the bursts (65%). The increased density of the top left quadrant in Fig. 6(b) is a clear indication of the gain obtained by trading a bit of speed for better accuracy. Finally, we stress that SWIFT *never* falls into the bottom right quadrant, irrespective of whether the historical model is used or not.

6.2.2 *Validation through simulation.*

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).
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]). When we triggered

	percentile of bursts						
	10th	20th	30th	50th	70th	80th	90th
Burst size between 2.5k and 15k							
CPR	24.6%	48.9%	72.6%	89.5%	98.5%	99.7%	99.9%
FPR	0%	0.03%	0.07%	0.22%	0.50%	0.81%	1.8%
CP	47	178	349	901	2.1k	3.0k	4.3k
FP	24	125	301	802	2.2k	3.1k	5.0k
Burst size greater than 15k							
CPR	5.6%	39.3%	80.4%	93.0%	98.1%	99.7%	99.9%
FPR	0%	0%	0.04%	0.60%	5.42%	13.9%	74.9%
CP	1.7k	5.7k	11.0k	19.6k	53.2k	78.1k	193k
FP	0	6	110	2.4k	19.8k	50k	402k

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

the inference at the end of the burst, SWIFT identified the failed link for 91% of the bursts (1991), 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 Appendix C).

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).

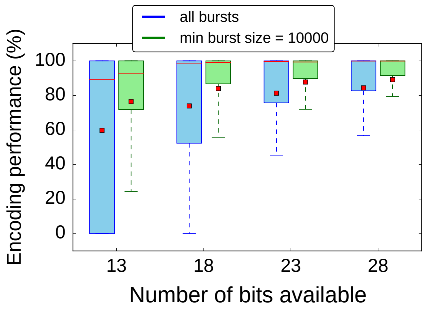


Figure 7: With only 18 bits available for the AS paths encoding, SWIFT can reroute more than 98.7% of the predicted prefixes in the median case.

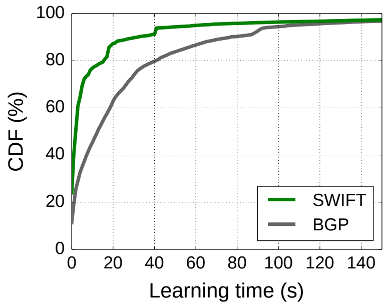


Figure 8: SWIFT quickly learns about remote outages. In 2 (resp. 9) seconds, SWIFT learns more than 50% (resp. 75%) of the withdrawals, BGP needs 13 seconds (resp. 32 seconds).

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 [46]. 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.

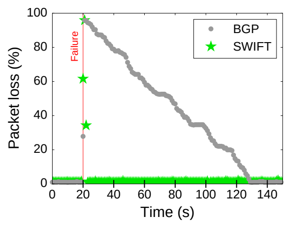
SWIFT requires few data-plane updates to reroute all the predicted prefixes. The number of data-plane updates required to reroute all the predicted prefixes depends on the number of failed AS links reported by the inference algorithm. When executing the inference algorithm after 2.5k withdrawals, in 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, 63], SWIFT can update all the forwarding entries within 130 ms.

7 CASE STUDY

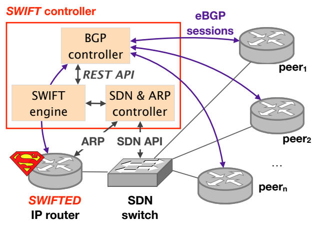
In this section, we showcase the benefits of SWIFT by boosting the convergence time of a recent Cisco router. As mentioned in §3.2, SWIFT can be implemented directly on existing routers via a simple software update, since the only hardware requirement, a two-stage forwarding table, is readily available 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).



(a) A SWIFTED Cisco router converge *98% faster.*



(b) Alternative SWIFT implementation on existing router.

Figure 9: While a recent router takes 110 seconds to converge upon a large remote outage (left), the corresponding SWIFTED router (using the alternative deployment scheme depicted on the right) converges within 2 seconds.

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 (OpenVSwitch 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, 35, 38, 39, 67–69], or within a network [20, 40, 58, 66]. 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 [35] or failures on the reverse path [38].

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 [33], 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, 43, 44, 47]. 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 [43] 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, 45, 48] 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 [51], IP fast reroute [11, 60] 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 [56], 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.

REFERENCES

[1] TCP Behavior of BGP. (2012). https://archive.psg.com/121009.nag-bgp-tcp.pdf.
[2] 5-minute outage costs Google $545,000 in revenue. (2013). http://venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue/.
[3] Cisco Systems. BGP PIC Edge and Core. (2015). http://www.cisco.com/c/en/us/td/docs/routers/7600/ios/15S/configuration/guide/7600_15_0s_book/BGP.html.
[4] Amazon.com went down for about 20 minutes, and the world freaked out. (2016). http://mashable.com/2016/03/10/amazon-is-down-2/.
[5] CIDR report. (2016). http://www.cidr-report.org/as2.0/.
[6] Cisco Umbrella 1 Million. (2016). https://blog.opendns.com/2016/12/14/cisco-umbrella-1-million/.
[7] ExaBGP. (2016). https://github.com/Exa-Networks/exabgp.
[8] Google cloud outage highlights more than just networking failure. (2016). http://bit.ly/1MFO2Ye.
[9] RIPE RIS Raw Data. (2016). https://www.ripe.net/data-tools/stats/ris/.
[10] Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. 2015. Hyperbolic graph generator. *Computer Physics Communications* (2015).
[11] A. Atlas and A. Zinin. Basic Specification for IP Fast Reroute: Loop-Free Alternates. RFC 5286. (Sept. 2008).
[12] Ritwik Banerjee, Abbas Razaghpanah, Luis Chiang, Akassh Mishra, Vyas Sekar, Yejin Choi, and Phillipa Gill. 2015. Internet Outages, the Eyewitness Accounts: Analysis of the Outages Mailing List.
[13] Zied Ben Houidi, Mickael Meulle, and Renata Teixeira. Understanding slow BGP routing table transfers. In *ACM IMC, 2009.*
[14] Anat Bremler-Barr, Edith Cohen, Haim Kaplan, and Yishay Mansour. 2002. Predicting and Bypassing End-to-end Internet Service Degradations. In *ACM SIGCOMM Workshop on Internet Measurment (IMW '02)*. ACM, New York, NY, USA.
[15] Matthew Caesar, Lakshminarayanan Subramanian, and Randy H Katz. 2003. *Towards localizing root causes of BGP dynamics.* University of California Berkeley.
[16] CAIDA. The CAIDA AS Relationships Dataset. (2016). http://www.caida.org/data/active/as-relationships/
[17] Jaideep Chandrashekar, Zhenhai Duan, Zhi-Li Zhang, and Jeff Krasky. Limiting path exploration in BGP. In *IEEE INFOCOM, 2005.*
[18] Di-Fa Chang, Ramesh Govindan, and John Heidemann. The Temporal and Topological Characteristics of BGP Path Changes. In *ICNP 2003.*
[19] Ítalo Cunha, Renata Teixeira, Darryl Veitch, and Christophe Diot. 2014. DTRACK: a system to predict and track internet path changes. *IEEE/ACM TON* (2014).
[20] G. Das, D. Papadimitriou, B. Puype, D. Colle, M. Pickavet, and P. Demeester. SRLG identification from time series analysis of link state data. In *COMSNETS, 2011.*
[21] Benoit Donnet and Olivier Bonaventure. 2001. On BGP communities. *ACM SIGCOMM CCR* (2001).
[22] Nick Feamster, David G. Andersen, Hari Balakrishnan, and M. Frans Kaashoek. Measuring the Effects of Internet Path Faults on Reactive Routing. In *ACM SIGMETRICS, 2003.*
[23] Anja Feldmann, Olaf Maennel, Z Morley Mao, Arthur Berger, and Bruce Maggs. 2004. Locating Internet routing instabilities. *ACM SIGCOMM CCR* (2004).
[24] Clarence Filsfils. BGP Convergence in much less than a second. (2007). Presentation NANOG 23.
[25] Clarence Filsfils, Pradosh Mohapatra, John Bettink, Pranav Dharwadkar, Peter De Vriendt, Yuri Tsier, Virginie Van Den Schrieck, Olivier Bonaventure, and Pierre Francois. 2011. *BGP Prefix Independent Convergence.* Technical Report. Cisco.
[26] Pierre Francois, Pierre-Alain Coste, Bruno Decraene, and Olivier Bonaventure. 2007. Avoiding disruptions during maintenance operations on BGP sessions. *IEEE Transactions on Network and Service Management* (2007).
[27] Lixin Gao. 2001. On inferring autonomous system relationships in the Internet. *IEEE/ACM TON* (2001).
[28] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *ACM SIGCOMM 2011.*
[29] Timothy G Griffin and Brian J Premore. An experimental analysis of BGP convergence time. In *IEEE ICNP, 2011.*
[30] Arpit Gupta, Robert MacDavid, Rüdiger Birkner, Marco Canini, Nick Feamster, Jennifer Rexford, and Laurent Vanbever. 2016. An industrial-scale software defined internet exchange point. In *USENIX NSDI 2016.*
[31] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. SDX: A Software Defined Internet eXchange. In *SIGCOMM 2014.*
[32] Nikola Gvozdiev, Brad Karp, Mark Handley, and others. LOUP: The Principles and Practice of Intra-Domain Route Dissemination. In *USENIX NSDI 2013.*
[33] Polly Huang, Anja Feldmann, and Walter Willinger. A non-instrusive, wavelet-based approach to detecting network performance problems. In *ACM SIGCOMM Workshop on Internet Measurement, 2001.*
[34] Gianluca Iannaccone, Chen-nee Chuah, Richard Mortier, Supratik Bhattacharyya, and Christophe Diot. Analysis of link failures in an IP backbone. In *ACM SIGCOMM Workshop on Internet measurement, 2002.*
[35] Umar Javed, Italo Cunha, David Choffnes, Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy. PoiRoot: Investigating the Root Cause of Interdomain Path Changes. In *ACM SIGCOMM, 2013.*
[36] John P John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson, and Arun Venkataramani. Consensus routing: The Internet as a distributed system. In *USENIX, 2008.*
[37] D. Katz and D. Ward. Bidirectional Forwarding Detection. RFC 5880. (2010).
[38] Ethan Katz-Bassett, Colin Scott, David R Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. 2012. LIFEGUARD: practical repair of persistent route failures. *ACM SIGCOMM CCR* (2012).
[39] Ravish Khosla, Sonia Fahmy, Y. Charlie Hu, and Jennifer Neville. 2011. Prediction Models for Long-term Internet Prefix Availability. *Computer Networks* (2011).
[40] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. IP fault localization via risk modeling. In *NSDI, 2005.*
[41] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. 2010. Hyperbolic Geometry of Complex Networks. *CoRR* abs/1006.5169 (2010). http://arxiv.org/abs/1006.5169
[42] Nate Kushman, Srikanth Kandula, and Dina Katabi. 2007. Can You Hear Me Now?!: It Must Be BGP. *ACM SIGCOMM CCR* (2007).
[43] Nate Kushman, Srikanth Kandula, Dina Katabi, and Bruce M Maggs. R-BGP: Staying connected in a connected world. In *USENIX NSDI, 2007.*
[44] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. 2000. Delayed Internet routing convergence. *ACM SIGCOMM CCR* (2000).
[45] Olaf Maennel and Anja Feldmann. Realistic BGP Traffic for Test Labs. In *ACM SIGCOMM, 2002.*
[46] Philipp Mao, Rudiger Birkner, Thomas Holterbach, and Laurent Vanbever. Boosting the BGP convergence in SDXes with SWIFT. In *ACM SIGCOMM, 2017 (Demo).*
[47] Z Morley Mao, Randy Bush, Timothy G Griffin, and Matthew Roughan. BGP beacons. In *ACM IMC, 2003.*
[48] Z Morley Mao, Ramesh Govindan, George Varghese, and Randy H Katz. Route flap damping exacerbates Internet routing convergence. In *SIGCOMM, 2002.*
[49] W.B. Norton. 2011. *The Internet Peering Playbook: Connecting to the Core of the Internet.* DrPeering Press.
[50] University of Oregon. Route Views Project. (2016). www.routeviews.org/.
[51] P. Pan, G. Swallow, and A. Atlas. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090. (May 2005).
[52] Vern Paxson. 2006. End-to-end Routing Behavior in the Internet. *ACM SIGCOMM CCR* (2006).
[53] Cristel Pelsser, Olaf Maennel, Pradosh Mohapatra, Randy Bush, and Keyur Patel. Route flap damping made usable. In *PAM, 2011.*
[54] Ponemon Institute. Cost of Data Center Outages. (2016). http://datacenterfrontier.com/white-paper/cost-data-center-outages/.
[55] B. Quoitin and S. Uhlig. 2005. Modeling the Routing of an Autonomous System with C-BGP. *IEEE Network Magazine of Global Internetworking* (2005).
[56] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. FatTire: Declarative Fault Tolerance for Software-defined Networks. In *HotSDN, 2013.*
[57] Y. Rekhter, T. Li, and S. Hares. 2006. *A Border Gateway Protocol 4.* RFC 4271.
[58] Matthew Roughan, Tim Griffin, Morley Mao, Albert Greenberg, and Brian Freeman. Combining Routing and Traffic Data for Detection of IP Forwarding Anomalies. In *SIGMETRICS, 2004.*
[59] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush. 2011. 10 Lessons from 10 Years of Measuring and Modeling the Internet's Autonomous Systems. *IEEE Journal on Selected Areas in Communications* (2011).
[60] M. Shand and S. Bryant. IP Fast Reroute Framework. RFC 5714. (Jan. 2010).
[61] Ashwin Sridharan, Sue B. Moon, and Christophe Diot. On the Correlation Between Route Dynamics and Routing Loops. In *ACM IMC, 2003.*
[62] Daniel Turner, Kirill Levchenko, Alex C. Snoeren, and Stefan Savage. California Fault Lines: Understanding the Causes and Impact of Network Failures. In *ACM SIGCOMM, 2010.*
[63] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *ACM SIGCOMM, 2015.*
[64] Stefano Vissicchio, Laurent Vanbever, Cristel Pelsser, Luca Cittadini, Pierre Francois, and Olivier Bonaventure. 2013. Improving Network Agility with Seamless BGP Reconfigurations. *IEEE/ACM TON* (2013).
[65] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A Measurement Study on the Impact of Routing Events on End-to-end Internet Path Performance. In *ACM SIGCOMM, 2006.*
[66] Junling Wang and Srihari Nelakuditi. IP fast reroute with failure inferencing. In *ACM SIGCOMM workshop on Internet network management, 2007.*
[67] Jian Wu, Zhuoqing Morley Mao, Jennifer Rexford, and Jia Wang. Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. In *USENIX NSDI, 2005.*
[68] Ying Zhang, Z Morley Mao, and Jia Wang. A framework for measuring and predicting the impact of routing changes. In *IEEE INFOCOM, 2007.*
[69] Ying Zhang, Z. Morley Mao, and Ming Zhang. Effective Diagnosis of Routing Disruptions from End Systems. In *USENIX NSDI, 2008.*

APPENDIX

A DETAILED INFERENCE ALGORITHM

Algorithm 1 depicts our implementation of the SWIFT link inference algorithm. The algorithm is launched during a burst of BGP withdrawals and takes as parameter G, the AS graph seen when the algorithm is launched, G_W the graph of withdrawn AS paths, and W_{total} the current size of the burst. G (resp. G_W) are weighted based on the number of prefixes traversing (resp. that traversed before the burst and are now withdrawn) their links. For each node in G_W, the algorithm sorts its ongoing links based on their individual FS score. Then, it repeatedly merges the outgoing links, starting from the one with the highest FS score, and recomputes the new FS score of this set of links. As soon as merging a new outgoing link decreases the FS score, the algorithm breaks (line 32) the loop and updates the set of links having the best FS score among all the visited nodes. With this break, the algorithm does not need to compute the FS score for all the possible combinations of links. Instead, few iterations are generally enough before processing another node in the graph. While this makes the algorithm fast, it also makes it non optimal. However, in the worse case scenario, it still returns the AS link with the highest individual FS score. For simplicity, we only show the merging of the outgoing links of each node. In our implementation, the algorithm also merges the FS score of the incoming links of each node.

Algorithm 1 SWIFT link inference algorithm

```
1:  G : the AS links topology
2:  G_W : the graph of withdrawn AS paths
3:  X.nb_prefixes(x, y) : returns the number of prefixes traversing
    the link $(x, y)$ in the graph $X$
4:  FS : computes the FS score for the given $W$, $P$ and $W_{total}$
5:  sort_based_on_FS : sorts a dictionary based on the FS score of
    each element for the given current size of the burst
6:
7:  function LINK INFERENCE($G$, $G_W$, $W_{total}$)
8:      $best\_FS = 0$
9:      $failed\_links = \emptyset$
10:     for $x$ in $G_W.nodes()$ do
11:         $L = dic()$
12:         for $y$ in $G_W.succ(x)$ do
13:             $W \leftarrow G_W$.nb_prefixes$(x, y)$
14:             $P \leftarrow G$.nb_prefixes$(x, y)$
15:             $L[y] \leftarrow (W, P)$
16:         $L$.sort_based_on_FS($W_{total}$)
17:
18:         $cur\_set \leftarrow \emptyset$
19:         $cur\_FS \leftarrow 0$
20:         $merged\_W, merged\_P \leftarrow 0$
21:         while $L$.length() $> 0$ do
22:             $tmp\_W, tmp\_P = L$.pop()
23:             $new\_W \leftarrow merged\_W + tmp\_W$
24:             $new\_P \leftarrow merged\_P + tmp\_P$
25:             $new\_FS$ = FS($new\_W, new\_P, W_{total}$)
26:             if $new\_FS > cur\_FS$ then
27:                 $cur\_FS = new\_FS$
28:                 $cur\_set.add((x, tmp\_ngh))$
29:                 $merged\_W \leftarrow new\_W$
30:                 $merged\_P \leftarrow new\_P$
31:             else
32:                 break
33:
34:         if $best\_FS < cur\_FS$ then
35:             $best\_FS \leftarrow cur\_FS$
36:             $failed\_links = cur\_set$
        return $failed\_links$
```

B GUARANTEES OF SWIFT

SWIFT is beneficial and safe. Despite not notifying path changes in the control-plane, we now show that the predictive rerouting strategy implemented by SWIFT is safe as long as inferences do not miss the disrupted resources and BGP paths do not arbitrarily change during rerouting.

Given an inter-domain outage O triggering SWIFT, safety conditions for predictive fast rerouting are formalized by the following assumptions.

- Stability assumption: During the period in which SWIFT fast reroutes traffic (i.e., before the end of BGP convergence), ASes change their inter-domain forwarding paths only to avoid O.
- Reasonable inference assumption: SWIFT inferences enable the SWIFTED routers to avoid a super-set of the links disrupted by O.

Under the reasonable inference assumption, the following lemma holds.

LEMMA B.1. *When any SWIFTED router fast reroutes, it sends packets over paths with no blackhole and loops.*

PROOF. Consider any SWIFTED router s that fast reroutes at a given time t, to avoid an inferred outage. Let n be the router to which s fast reroutes. By definition of SWIFT (and any existing fast rerouting technique that s can apply if next to a disrupted link), the following properties hold at t:

- n must offer a BGP path P_n to s, otherwise s would have not fast rerouted to n. By definition of BGP, P_n does not contain loops.
- P_n must not include any of the links affected by the outage, *i.e.,* it does not include blackholes. This is a direct consequence of the reasonable inference assumption.
- n and all routers in P_n keep forwarding packets over P_n. This is always true for routers that do not fast reroute (using SWIFT or any other fast rerouting technique), by definition of BGP. Also, since P_n does not include any link affected by the outage (see previous property), routers in P_n that can fast reroute do not receive any withdrawal for path P_n (nor any path update, since SWIFT and other fast rerouting techniques do not generate BGP messages). As a consequence, they all also maintain P_n as forwarding path.

Combining these properties together, the packets fast rerouted by s at time t are forwarded over the path $(s\ n) \cup P_n$, which does not contain loops nor blackholes – which proves the statement. □

This lemma leads to the following theorems.

Theorem 3.1. *Under the stability and reasonable inference assumptions, the number of disrupted paths is decreased by every SWIFTED router which is on a path impacted by an outage.*

Proof. The statement follows by observing that every SWIFTED router on a disrupted path (i) will fast reroute, if the reasonable inference assumption holds, and (ii) will redirect traffic over a non-disrupted path by Lemma B.1. □

Theorem 3.2. *Under the stability and reasonable inference assumptions, no SWIFT rerouting causes any forwarding loop, irrespectively of the set of SWIFTED routers.*

Proof. Assume by contradiction that upon an outage (affecting one or more inter-domain links) a forwarding path for a certain prefix contains a loop L at a given time during the BGP convergence. Since BGP is guaranteed to compute non-loopy paths, at least one router s in L must fast reroute. However, s cannot fast reroute to a path including a loop, by Lemma B.1. This contradicts the hypothesis, and yields the statement. □

SWIFT inference is sound. We now prove Theorem 4.1. For the sake of simplicity, we implicitly assume that the inference input is produced by a single neighbor of the SWIFTED router. However, the proof is easy to extend since inference algorithm performs all its operations (including metric computation) on a per-neighbor basis.

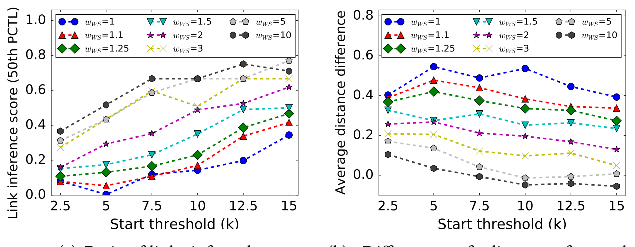
Theorem 4.1. *If all ASes inject at least one prefix on every adjacent link, SWIFT inference returns a set of links including the failed link if run on the corresponding stream of BGP messages.*

Proof. Assume that a single link f fails and that the inference algorithm makes a prediction at time t when fed with all and only the BGP messages generated by f.

We now show that the inference algorithm assigns the highest possible values of both WS and PS to f.

Indeed, all the paths traversing f before the burst are either explicitly withdrawn or updated (to avoid f): This implies that the number $P(f,t)$ of paths traversing f at t is 0. Moreover, since only BGP messages generated by f's failure are in the inference input by hypothesis, all the received withdrawals must have crossed f, that is, $W(f,t) = W(t)$. As a consequence, $PS(f,t) = W(l,t)/(W(l,t) + 0)$ and $WS(f,t) = W(l,t)/W(l,t)$ are equal to their maximum value 1.

This implies that the fit score of f is the highest possible one, hence the SWIFT inference algorithm will return it in the set of failed links. □



(a) Ratio of links inferred.

(b) Difference of distance from the SWIFTED router.

Figure 10: The inference of failed links improves when it is executed later or when the weight on the WS is higher.

C INFERENCE ALGORITHM CALIBRATION

Two parameters helps adapting the behavior of SWIFT inference algorithm so that it is more robust against real world factors: *(i)* the *start threshold*, that is, the number of withdrawals after which SWIFT inference is triggered; and *(ii)* the difference between w_{WS} and w_{PS} (see §4.1). We now discuss how we calibrated these parameters, by studying how they affect the performance of the link failure localization.

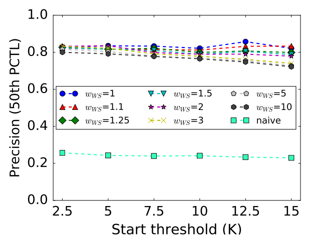
To ensure that our results are not dataset-driven, we rely a different dataset than the one we use in the evaluation (§6). Specifically, we rely on 375 bursts greater than 20k withdrawals collected on 10 RouteViews collectors during the last two weeks of July 2016. As we do not have ground truth, we compare results obtained when we run the algorithm with different values of the start threshold with respect to the values obtained at the end of each burst (*i.e.,* with complete information). We focus first on calibrating the *start treshold* before calibrating the weights.

A start treshold of only 2.5k withdrawals is enough to guarantee good inference performance. Figure 10(a) shows the (median) ratio of links inferred at a given start threshold with respect to the ones inferred at the end of the burst and this, for different weight values. Intuitively, we can see that results converge towards 1 as the start threshold increases.

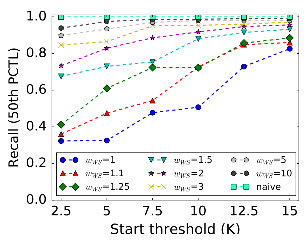
We can see that with weights $w_{WS} \geq 3$, the inference score is between 0.3 and 0.4 after 2.5k withdrawals, and between 0.7 and 0.8 after 15k withdrawals. These results reveal that after 2.5k withdrawals, SWIFT already localizes (sometimes partially) many outages.

As a result, we configure SWIFT to run the inference algorithm after only 2.5k withdrawals. At the same time, we configure SWIFT to wait for more information if the number of prefixes that would end up being rerouted is unlikely (based on a historical model, as explained in §4.2). Specifically, we reject an inference that would reroute more than 10k, 20k, 50k and 100k prefixes, if respectively inferred after 2.5k, 5k, 7.5k and 10k withdrawals.

Increasing w_{WS} leads to better performance. Figure 10(a) shows that increasing the weight of WS (while keeping the weight of PS equal to 1) can improve the inference. Figure 10(b) helps understanding this behavior by showing the difference, in the distance from the SWIFTED router, between the links inferred before and at the end of each burst. With equal weights for WS and PS, the



(a) Precision.



(b) Recall.

Figure 11: Sensitivity analysis of prefix prediction.

algorithm tends to infer links further than the one inferred at the end of the burst. This is likely because at the beginning of the burst, the PS of the failed link is lower than 1 (as explained in §4.2), and thus further links may have a higher PS, since less prefixes are using them. To bring the distance difference closer to 0 (ideal case, if we assume that the inference at the end of the burst is the correct one), we can increase the weight of WS. Figure 10(b) shows that when choosing 3, 5 or 10 for the WS weight, the average distance difference is very close to 0, and can even be negative for 5 and 10. As a result, we set the ratio between the weights of WS and PS to 3.

The accuracy of the link inference drives the performance of the withdrawals prediction. When tuning the start threshold and the w_{WS} to improve the link inference, this in turn also improves the withdrawals prediction. We evaluate the accuracy of the withdrawals prediction (in terms of *precision* and *recall*) against the actual withdrawals received at the end of the burst (these withdrawals act as ground truth in our dataset). For each burst in our dataset, we run the inference algorithm after a given number of withdrawals (start threshold) and with different w_{WS}, and we compare the future withdrawals predicted by the algorithm with the remaining ones in the burst. Figure 11 shows that the value of the *recall* can greatly improves (Fig. 11(b)) when increasing the weight of WS. On the other hand, increasing the weight of WS negatively impacts *precision* (11(a)), but with a much lower magnitude than the positive effect seen on recall. In contrast, a naive solution consisting in rerouting all the prefixes of the BGP session would have a recall equal to one but a very low precision.