The Next Generation of BGP Data Collection Platforms

Thomas Alfroy*, Thomas Holterbach*,
Thomas Krenc†, KC Claffy†, Cristel Pelsser*‡

*University of Strasbourg, †CAIDA/UC San Diego, ‡UCLouvain

bgproutes.io

ABSTRACT

BGP data collection platforms as currently architected face fundamental challenges that threaten their long-term sustainability. Inspired by recent work, we analyze, prototype, and evaluate a new optimization paradigm for BGP collection. Our system scales data collection with two components: analyzing redundancy between BGP updates and using it to optimize sampling of the incoming streams of BGP data. An appropriate definition of redundancy across updates depends on the analysis objective. Our contributions include: a survey, measurements, and simulations to demonstrate the limitations of current systems; a general framework and algorithms to assess and remove redundancy in BGP observations; and quantitative analysis of the benefit of our approach in terms of accuracy and coverage for several canonical BGP routing analyses such as hijack detection and topology mapping. Finally, we implement and deploy a new BGP peering collection system that automates peering expansion using our redundancy analytics, which provides a path forward for more thorough evaluation of this approach.

CCS CONCEPTS

• **Networks** → **Network measurement**.

KEYWORDS

Internet measurement, BGP, Routing Security

ACM Reference Format:

Thomas Alfroy, Thomas Holterbach, Thomas Krenc, KC Claffy, Cristel Pelsser. 2024. The Next Generation of BGP Data Collection Platforms. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3651890.3672251

1 INTRODUCTION

The study of the global Internet infrastructure relies on BGP data collection platforms (RouteViews [61] and RIPE RIS [49]) that maintain BGP peering sessions with network operators who volunteer to share (sometimes portions of) their routing tables. Originally established decades ago to support operational troubleshooting ("How do others reach my network?"), these systems have become a cornerstone for scientific and operational analysis of the Internet.

Collecting this data faces a fundamental cost-benefit trade-off. The information-hiding character of BGP requires collecting routes from as many BGP routers, *a.k.a* Vantage Points (VPs), as possible. But in practice the BGP protocol extensively propagates connectivity messages, leading to highly redundant (along with significant unique) data coming from each peer. The result is a data set with enormous redundancy and yet dangerous visibility gaps [34].

The platforms' policies to store a snapshot of the aggregated data every few hours, as well as every BGP update received in between these snapshots, exacerbates the storage of redundant data. Continued growth of the Internet (≈ 75k ASes [14] and ≈ 1M globally announced prefixes) and increasing connectivity between networks further burden data collection and use [1, 28]. Users often resort to sampling the data, e.g., using only a sample of the VPs, neglecting the connectivity uniquely visible to other VPs. Finally, the manual vetting of new peers also strains platform scalability. The platforms collectively peer with only ≈1% of the observably active ASes on the global Internet. Despite continued addition of peers, RIS and RV's coverage in terms of fraction of ASes they are peering with has remained flat for two decades.

These growing pressures coincide with regulatory concerns about slow progress in deployment of routing security protections [62]. The ensuing public debate has highlighted the importance of these platforms for detecting both accidental and malicious transgressions in the routing system. While significant investment in data collection could accommodate gathering, retention, and sharing orders of magnitude more routing data, current constraints motivate us to consider a more strategic approach. We propose a data collection scheme that scales at least an order of magnitude in the number of VPs feeding public collection systems while limiting the increase in human effort and data volume.

Vision. We explore a fundamentally new way to collect BGP data: an *overshoot-and-discard* strategy. Akin to CERN's Large Hadron Collider (LHC) which generates millions of collisions just to see a few interesting particles (e.g., Higgs boson), overshooting BGP data collection will maximize the chance to see interesting routing events, e.g., BGP hijacks. We imagine a world where public BGP data providers could automate deployment of additional VPs, targeting a *moonshot* of peering with one VP in every of the ≈75K ASes participating in the global routing system (even half would be a moonshot!). Overshooting BGP data collection is feasible only if the system can discard the "less interesting" bits upon acquisition,

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672251

before they consumes processing or storage resources. In the case of the LHC, fast online algorithms using custom hardware and software discard 99.994% of the likely less interesting collisions [55].

The epistemological challenge is that *discarding BGP updates inevitably implies loss of information*. Even if two VPs observe the identical (or similar) prefix announcement, there is a signal in knowing which VPs observed it. We construct a framework that allows some context-dependence in the definition of redundancy and apply it to several important use cases for BGP data. For example, a BGP hijack may reach many VPs, which redundantly observe it, or it may reach no available VP (perhaps by intention [34]), in which case substantial expansion of VP deployment is the best way to increase the chance of observing it.

Contributions. We make the following contributions.

- We demonstrate with a survey, measurements, and simulations that the RIS and RV coverage limitations limit effective scientific use of the data. (§2-§3).
- We characterize redundancy in BGP data, design algorithms that identify redundant updates and VPs for flexible redundancy definitions, and implement *GILL*, a BGP data collection system that uses an *overshoot-and-discard* collection scheme (§4-§5-§6-§7).
- We deployed *GILL* at https://bgproutes.io and publish all its data. Operators can automatically connect their BGP routers to *GILL* to contribute BGP data (§8-§9).
- We show that *GILL*'s sampling algorithms outperform the status quo for five relevant use cases (§10).

Long-term impact (§11). *GILL*'s approach offers a long-term path toward sustainable scaling of BGP data collection. We show that a redundancy-aware system consistently improves the accuracy and coverage of studies and tools that rely on BGP data. Our simulations of a scenario where 50% (vs. 2%) of ASes peered with *GILL* tripled the number of peer-to-peer links observed, doubled the number of Internet failures that we could localize, and reduced by 33% the proportion of undetected forged-origin hijacks without processing more data than what RIS and RV do today.

Immediate benefits (§12). Regardless of the future of the *GILL* platform, its sampling algorithms can help users cope with the massive stream of data that RIS and RV generate. We replicated analyses in three studies/tools, in all cases *GILL* improved the accuracy and coverage while processing the same data volume: we inferred more AS relationships (+16%), identified and corrected errors in CAIDA's ASrank dataset, and inferred more forged-origin hijacks (+23%) with ≈4× fewer incorrect inferences (i.e., false positives).

2 BACKGROUND

Routing Information Service (RIS) [49] and RouteViews (RV) [61] are two widely-used platforms that peer with hundreds of routers (also called peers, or VPs) and collect the BGP updates exported by those VPs. As of May 2023, 32% of the RIS and RV VPs [37, 52] are *full feeders*, i.e., they send updates for roughly all announced IP prefixes on the Internet (≈944k IPv4 and ≈205k IPv6 prefixes [14]). A stored BGP update carries four relevant attributes [40]: *(i)* the timestamp at which the update was received, *(ii)* the IP (v4 or v6) prefix that the update announces, *(iii)* the AS path used to reach that prefix, and *(iv)* a set of BGP communities, which carry information or requests for special handling of the announcement. Among other uses, researchers leverage the timestamp to find transient paths [30], the prefix to detect hijacks [56], the AS paths to infer AS relationships [31], and the communities to understand complex routing behavior [29, 60].

The path-vector nature of BGP challenges macroscopic data collection efforts, because each router only announces updates for its best route to each destination, limiting the visibility of backup links. Confidential routing policies also limit the propagation of updates and thus visibility of links. Thus, each VP (even full feeders) provides a *partial view* over Internet routing. We illustrate in Fig. 1 the inherent limitation in mapping AS topology by combining partial views. Assume that every AS runs a single BGP router, announces its only prefix into BGP, and configures routing policies following the Gao-Rexford model [23]. Straight lines are customer-to-provider (c2p) links and dashed lines are peer-to-peer (p2p) links. Logically, VPs at the core help to observe c2p links whereas the ones at the edge help to observe p2p links (as they are not announced to providers [23]). With the local view of 1, one can infer all AS links but the two peering links 3—4 and 5—6 (Fig. 1a), whereas with the local view of 5, one can infer all AS links but the two customer-to-provider links 2—4, 4—6 (Fig. 1b). Local views can be redundant, e.g., combining local views of 1 and 2 does not reveal more links (Fig. 1c).

To expand coverage, RIS and RV continually add new VPs; by Dec 2023, RIS had 1537 VPs in 816 distinct ASes and RV had 1130 VPs in 337 distinct ASes (Fig. 2, top). But the total number of active ASes on the Internet grows faster than the platforms' peering expansion, so the net *coverage*, i.e., the proportion of ASes that host at least one VP, is stable (Fig. 2, bottom). Users can download a snapshot of all BGP updates held by a VP at a particular time also called a *routing table* or *RIB (Routing Information Base)*. Alternatively, users may download every update observed by the VPs over time (e.g., using [40]), which currently results in ≈28K updates per hour (average in Dec. 2023) for a single VP (Fig. 3a), and billions of updates per day for all RIS and RV VPs (Fig. 3b).

3 INCREASING COVERAGE

We use three case studies—AS topology mapping, locating outages, and BGP hijack detection—to demonstrate how expanding these platforms to support more VPs would improve the accuracy and coverage of scientific and operational analyses of Internet infrastructure (§3.1). We explain the challenges of such expansion for data providers and users (§3.2).

3.1 Limitations of low VP coverage

A tiny fraction (1.1%) of the 74k ASes participating in the global routing system [14] host a VP. If we consider only the 11832 transit ASes (i.e., those with at least one customer), this fraction is higher but still only 5.9%. While we cannot know how much additional information we might observe from VPs that do not peer with the public collection platforms, we estimate this gap using controlled simulations. We use C-BGP [47] to simulate "mini" Internets where each AS runs one BGP router and announces one or more prefixes.

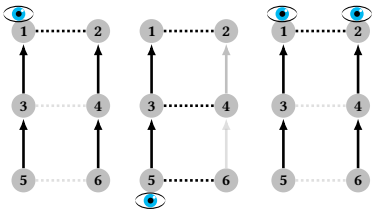


Figure 1: Combining local views can help to map the AS topology. Gray links are not visible from routes collected by VPs (👁).

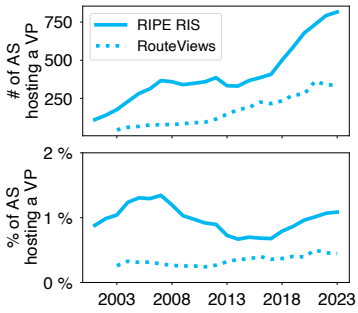
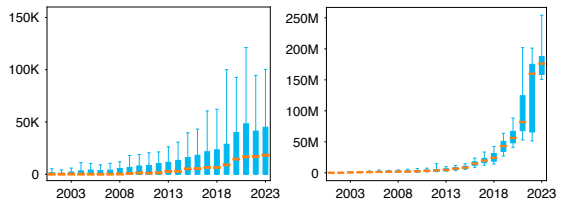


Figure 2: Growth in VPs.



(a) Hourly average # of updates per VP. **(b) Updates per hour among all VPs.**

Figure 3: Growth in updates collected by RIS and RV combined.

We ensure that the number of prefixes announced by the ASes follows the distribution observed in the real Internet. We then collect routes from a subset of ASes selected randomly and measure how well they enable achieving various objectives.

Used AS topologies. We run our simulations on eleven ASes topologies generated using two techniques.

Pruned known AS topology: We infer the AS topology from CAIDA's AS relationship dataset from October 2023 [19] and prune it (to reduce the computational/hardware cost of the simulations) by iteratively removing leaf nodes until the topology has 6k ASes (or 1k depending on the objective). While we cannot scale our simulations to the size of the real Internet, we note that they are larger than simulations conducted in previous studies [26, 35, 36].

Artificial topologies: We built ten AS topologies whose statistical parameters match those of the known AS topology using the Hyperbolic Graph Generator [3]. We set the average node degree to 6.1, which results in a comparable degree of connectivity (*a.k.a.* Beta index) to the one observed in CAIDA's AS relationship dataset [19], and use as the degree distribution a power law with exponent 2.1 (as in [3]). We assign AS relationships as follows. The three ASes with the highest degree are Tier1s and fully meshed. ASes connected to a Tier1 are Tier2s. ASes connected to a Tier2 but not to a Tier1 are Tier3s, etc. Two connected ASes have a p2p relationship if they are on the same level, and a c2p relationship if not. Routing policies follow the Gao-Rexford model [23].

Studied objectives. We use these topologies to estimate the impairment in our ability to perform three canonical inferences: AS topology mapping, link failure localization, and forged-origin hijack detection.

AS topology mapping: We measure the proportion of p2p and c2p links observed in at least one collected AS path. We consider p2p links separately since routing policies typically reduce their propagation and thus observability [23].

Link failure localization: We simulate 1k random link failures and measure how many p2p and c2p links we can locate using the algorithm described in [21]. Here, we use a topology with 1k ASes (instead of 6k for the other objectives) as this analysis is more computationally expensive.

Forged-origin hijack detection: In these hijacks, the attacker prepends the valid origin in the AS path [25]. Type-X hijacks are forged-origin hijacks where $X \geq 1$ is the position of the attacker's AS in the forged AS path. We simulate a Type-1 and a Type-2 hijack for every possible victim and measure how many we detect from the collected routes. Attackers are randomly picked and hijack one of their victim's prefixes.

Observations. Fig. 4 shows the percentage of observed AS links (bottom), localized failures (middle), and detected forged-origin hijacks (top) as a function of the number of ASes hosting a VP (coverage). The results of the simulations with the pruned known AS topology are indicated with a star whereas the results from the ten artificial topologies are shown in boxes. We make two key observations.

Key observation #1: **The simulations effectively illustrate the opportunity cost of having only a 1% coverage of VPs**, i.e., the coverage of RIS and RV combined.

AS topology mapping: With so few VPs, simulations observed only 16% (resp. 12%) of the p2p links (median) when using the artificial topologies (resp. pruned known AS topology).

Failure localization: Only 10% (median of the ten simulations) of the failures on p2p links can be localized. With the pruned known AS topology, even c2p link failures are difficult to localize: with 1% coverage, we locate only ≈40% of them.

Hijack detection: With a 1% coverage, we fail to detect 24% (median) of Type-1 hijacks, i.e., they are not visible from *any* VP when using artificial topologies (16% when using the known pruned topology). Type-2 hijacks are even less visible (32% with a 1% coverage) because the hijacked routes have a longer AS path. The implication is that forged-origin hijack detection systems [25, 56] miss a significant fraction of hijacks, even if using all RIS and RV VPs. Given the prevalent use of these platforms for hijack detection, their lack of coverage leaves open significant attack surface [34].

Key observation #2: Our simulations suggest that the percentage of ASes hosting a VP should grow by 25-100× to achieve the three objectives reasonably well. With 50% of ASes hosting a VP (i.e., a 50× coverage increase), 90% of p2p links are mapped, 95% of failures on p2p links can be localized, and only 4% of Type-1 hijacks remain undetected (median values with artificial topologies).

Confirmation with real (but private) data. We contacted a private BGP data provider (bgp.tools [17]) that collects BGP routes from ≈1000 routers and compared the set of AS links observed from these private feeds against the set observed by RIS and RV VPs (in September 2023). We found that bgp.tools saw 192k AS links that *none* of the RIS and RV VPs observed, and conversely, RIS and RV VPs observed 401k links that bgp.tools did not observe. Other private data collection systems, e.g., that companies support, have reported visibility not seen in the public systems [12]. These

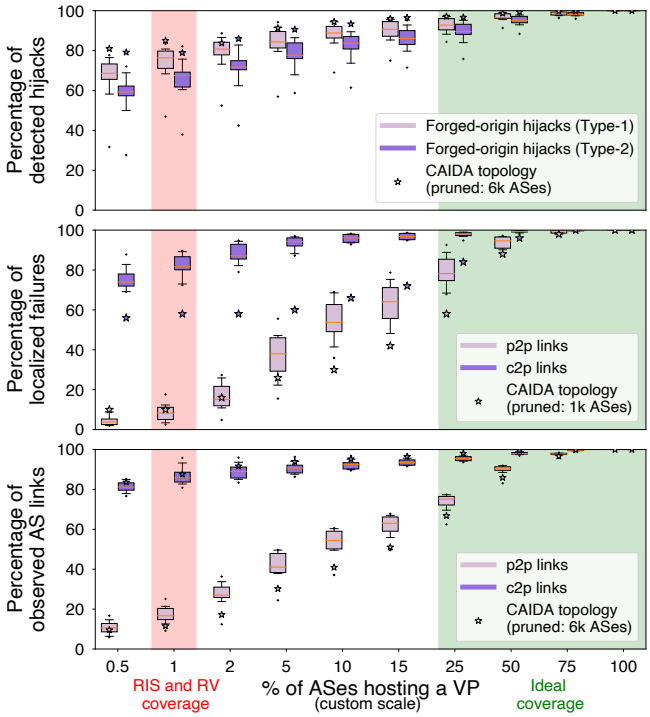


Figure 4: Our simulations show that the current RIS and RV coverage (1.1%, the red area) induces a significant impairment to important operational analyses. We suggest a 25-100× higher coverage (green area).

significant differences in visibility across VPs provide compelling motivation to architect data collection systems that can easily support many more VPs.

3.2 Scaling challenges in data collection

Putting aside the non-technical challenges of a radical expansion in the number of peers, we focus first on the technical challenges, for both data providers and users.

Challenges for data providers Cultivating more VPs generates more data as each of them exports BGP updates. Moreover, new IP prefixes advertised in BGP [14] contribute to the increase in the volume of data collected by every VP. The compound effect—more VPs (Fig. 2, top) and more updates per VP (Fig. 3a)—yields a *quadratic* increase in updates reaching the collection platforms (visible in Fig. 3b). RIPE RIS has expressed concerns about this unsustainable growth rate [1] and its implications for long-term data management [28]. Recently, RIPE removed the peering form (that network operators used to submit peering requests) from their website, to limit data processing and storage costs. Instead they adopted a selective peering policy, targeting peers in countries with the largest inferred visibility gaps.

Challenges for users (survey-based). Although several tools can speed up data processing [5, 6, 40], many measurement studies and monitoring tools use only a sample of data collected by RIS and RV, either using only a subset of the VPs, a short time window, or both. While authors of these studies do not typically explain why they sample, their choice suggests they believe the data volume is not worth trying to manage. We confirm this explanation with a survey we conducted on authors of 11 research papers. We do not cite these papers to preserve the anonymity of the respondents.

More precisely, we selected 11 BGP-based papers from top conferences, namely SIGCOMM, NSDI, S&P, USENIX Security, NDSS and IMC. We focused on studies published fewer than ten years ago that collectively covered a wide range of BGP-related questions. We purposely did not select any of the studies used to benchmark and evaluate *GILL*'s algorithm (§10-§11-§12). We classified these 11 studies into two categories based on how they sampled the BGP data. Nine papers used **all** routes collected from a **subset** of the VPs (which we call category C_1); six papers used a **limited durations** (C_2). Note that a paper may be in both categories. For each paper, we asked authors whether BGP data volume limited their work, how and why they sampled BGP data sources, their understanding of the impact of the sampling on the quality of their results, and if they would expand their sample given more resources or time. We did not receive answers from the authors of three papers. Thus, we have seven respondents in C_1 and five in C_2. We summarize our findings here; details of the survey are in an appendix (§16).

Key observation #1: The volume of BGP data to process is often a limiting factor. In fact, seven (of eight) respondents found the BGP data expensive to process. For three respondents in C_1, processing time motivated them to use only a subset of the VPs; three respondents in C_2 considered the processing time when choosing a measurement interval. Even a respondent who used a Spark cluster found it inhibitively time-consuming to process the BGP data.

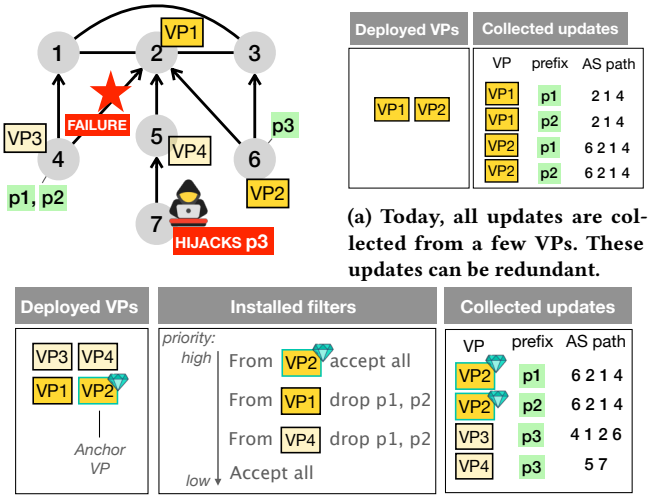
Key observation #2: Users often sacrifice quality of the results to facilitate data processing. In fact, six respondents in C_1 acknowledged that using more VPs would improve the quality of their analysis. The last respondent was not sure, given the potential redundancy in the data sources (which he did not analyze). Two of the six believed it would not *significantly* change the conclusion of their studies (e.g., one said that it could help to pinpoint corner cases). However, six of the seven authors in C_1 affirmed that they would have used more VPs if they had more resources and time. Similarly, all five respondents in C_2 said that extending the duration of their study would improve the quality of their results. One respondent thought the gain would not be significant; another said it could help detect rare routing events. All respondents in C_2 would have extended the duration of their observation window given more time and resources. The uncomfortable truth is that we do not know exactly what they are missing, which is why we used simulations (§3.1) and experiments (§12) to corroborate that important analyses lose accuracy and/or coverage when using heavily sampled topologies.

4 REDUNDANCY IN BGP DATA

We show, both intuitively (§4.1) and experimentally (§4.2), that redundancy in BGP data makes it a good candidate for collection with an overshoot-and-discard strategy.

4.1 Motivating example

Consider the scenario in Fig. 5 that shows seven ASes (1-7) interconnected in c2p (arrows) and p2p (lines) relationships, according to



(a) Today, all updates are collected from a few VPs. These updates can be redundant.

(b) *GILL* collects fewer but less redundant updates that enable inferring the failure in both directions and detecting the hijack.

Figure 5: A scenario showing the value of an overshoot-and-discard approach when collecting BGP data.

the Gao-Rexford model [23]. *AS4* originates two prefixes p1 and p2 whereas *AS6* originates one prefix, p3. Four VPs (1-4) export their routes to a data provider like RIS or RV. We consider two isolated events: (1) an Internet link failure impacting the peering session between *AS2* and *AS4*, and (2) a hijack where *AS7* illegitimately announces p3, the prefix owned by *AS6*. We consider only the updates induced by these two events.

Strong data redundancy at different granularities. Assume a deployment of only two VPs: VP1, VP2. In this case, four updates are collected (Fig. 5a), induced by the link failure causing a path change via 1 (the hijack is not visible from these VPs). We observe data redundancy at two levels of granularity: among collected updates and between VPs.

Redundancy between updates: Updates for p1 and p2 are redundant, induced by the same event (the link failure) and with similar attribute values (time and AS path).

Redundancy between VPs: VP1 and VP2 are redundant: They provide a very similar view over routing updates. For instance, they both receive an update for p1 and p2 at roughly the same time and with a similar AS path.

4.2 Exploring redundancy in the BGP data

We now provide a comprehensive analysis of redundancy in the BGP data. As there is no consensus on how to define redundant BGP updates, we define three gradually stricter definitions of redundancy between updates.

We denote $u(v, t, p, L, L_w, C, C_w)$ a BGP update observed by VP v at time t for prefix p. L is the set of AS links in the AS path whereas L_w is the set of AS links in the AS path implicitly withdrawn by this update, i.e., that were in the previous update for prefix p and are rendered obsolete by the new update. Similarly, C is the set of community values and C_w is the set of community values implicitly withdrawn for prefix p. Observe that $L_w = C_w = \emptyset$ if there was no previous update for p observed by v. Consider two BGP updates $u_1(v1, t1, p1, L1, L1_w, C1, C1_w)$ and $u_2(v2, t2, p2, L2, L2_w, C2, C2_w)$. We define the following three conditions to support our redundancy definitions:

- Condition 1: $|t_1 - t_2| < 100$s, and $p_1 = p_2$
- Condition 2: $L1 \setminus L1_w \subset L2 \setminus L2_w$
- Condition 3: $C1 \setminus C1_w \subset C2 \setminus C2_w$

Condition 1 uses a 100-seconds slack when comparing timestamps to accommodate typical BGP convergence time [30]. Condition 2 checks whether the set of new links in the AS path observed by VP $v1$ is included in the set of new links in the AS path observed by VP $v2$. Condition 3 follows the same approach but for community values. Observe that conditions 2 and 3 are asymmetric ($X \subset Y \implies Y \subset X$). We formalize our three gradually stricter redundancy definitions:

- Definition 1 *(prefix based)*:
  u_1 is redundant with u_2 if condition 1 is true
- Definition 2 *(prefix and AS-path based)*:
  u_1 is redundant with u_2 if conditions 1 and 2 are true
- Definition 3 *(prefix, AS-path and community based)*:
  u_1 is redundant with u_2 if conditions 1, 2 and 3 are true

The vast majority of the collected updates are redundant with another collected update. Among the updates collected by RIS and RV during one hour in Sept. 1st 2023, we find that 97% are redundant with at least another update according to Def. 1. This number remains high with stricter redundancy definitions (77% with Def. 2 and 70% with Def. 3).

A significant portion of the VPs are redundant with another VP. We use our redundancy definitions to quantify redundancy between RIS and RV VPs. We define VP_1 as redundant with VP_2 if >90% of the updates from VP_1 are redundant (based on one of the three definitions) with at least one update from VP_2. Fig. 6 shows the redundancy, for each of the three definitions, between 100 VPs randomly selected and computed using one hour of data on Sept. 1st, 2023. We focus on 100 VPs to reduce the computational resources needed to perform the experiment. However, we mitigate possible biases induced by this sampling (see §3) by performing 30 random selections with different seeds and showing the results for the selection that returns the median number of redundant pairs of VPs. With Def.1, 70% of the VPs are redundant with at least another VP. Logically, this number decreases with stricter definitions but remains significant: With Def. 2, 26% of the VPs are redundant with another, and 22% with Def. 3. We observe similar redundancy when considering only full feeders.

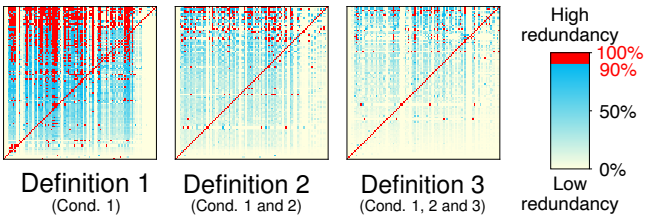


Figure 6: Redundancy among 100 random RIS and RV VPs for three gradually stricter redundancy definitions.

5 *GILL*'S KEY PRINCIPLES

We present *GILL*, a system that scales to as many ASes as wish to peer with it, to increase coverage while keeping data collection and processing manageable. We explain *GILL*'s two key principles: an overshoot-and-discard collection scheme and support for a flexible definition of *redundant*.

Overshoot-and-discard data collection strategy. *GILL* collects data using an *overshoot-and-discard* approach. "*Overshoot*" means that *GILL* can peer with tens of thousands of VPs (25-100× more than RIS and RV) while "*discard*" means that the redundant bits of the data are discarded right away to facilitate storage and processing. We illustrate this strategy using the scenario in Fig. 5. Assume the following three filters are applied to receive updates from four VPs:

from VP1 drop updates for prefixes p1, p2;
from VP2 drop updates for prefix p1;
from VP4 drop updates for prefix p1, p2;

Note that the second filter (for VP2) is not in Fig. 5, as a filter explained in the next paragraph (ingredient #2) overrides it. These filters retain three updates:

VP2 receives update for p2 with AS path 6—2—1—4;
VP3 receives update for p3 with AS path 4—1—2—6;
VP4 receives update for p3 with AS path 5—7;

These three updates provide richer information than the original four updates enabling more complete inferences. In fact, the update received by VP2 enables detection that one direction of link 2—4 is not used, and the update received by VP3 enables detection that the other direction is not used. Moreover, VP4 is close to the hijacker and observes the hijacked route, which is preferred over the legitimate ones in this region of the topology. Once collected, this hijacked route enables monitoring systems to detect the hijack and report it to the victim. This scenario demonstrates the possibility of gathering more insight from less but intelligently filtered BGP data, using more VPs.

We intentionally placed additional VPs (VP3 and VP4) and optimized filters to detect the two routing events and discard updates with similar attribute values. For example, the four updates that VP1 and VP2 observe for p1 and p2 have a similar AS path; *GILL* retains only one of these updates. In practice, deciding which updates to discard and building filters is challenging. There is no ground truth about which routing events will appear where, how they will propagate, and what users want to do with the data.

Sampling algorithms that maximize fairness. Our design objective is a general framework that is beneficial regardless of what users do with the data. However, discarding data inevitably affects some studies more than others. Maximizing fairness is challenging, especially given the diverse objectives that operators and researchers may have. *GILL* relies on a new sampling scheme that uses two key ingredients to maximize fairness.

Ingredient #1: Support for a flexible definition of redundant. While the three redundancy definitions in §4 enable us to illustrate redundancy across BGP updates and VPs, optimizing our algorithms to minimize redundancy according to a definition leads to overfitting. We explore this risk of overfitting by developing three **specific** BGP data sampling strategies, *each optimized to minimize redundancy in the set of updates collected by a sample of VPs according to one of the definitions in §4.* These three specific sampling strategies greedily select the VP that minimizes the proportion of collected redundant updates. Logically, a specific sampling strategy returns less redundant VPs compared to selecting them randomly (as in Fig. 6) when redundancy is evaluated according to the definition it is optimized for. For instance, when we use the specific sampling strategy that uses the loose redundancy Def. 1 to sample 100 RIS and RV VPs, only 37 VPs have >50% of their updates that are redundant with the ones observed by another VP. This number drops to 20 with Def. 2 and 15 with Def. 3. However, we benchmarked these specific sampling strategies on various use cases (e.g., hijack detection) and found that they perform poorly (§10).

Thus, we designed *GILL*'s sampling algorithm to not optimize for a given objective. Instead, *GILL* finds correlations across past BGP updates and uses a metric called *reconstitution power* to identify updates to discard because they can be inferred from other updates. *GILL* retains the latter updates.

Ingredient #2: Keep all updates from a few valuable VPs. Some studies require data for all prefixes (even if redundant), which is the case when one wants to identify the origin AS of every prefix. Ingredient #1 does not ensure visibility over all prefixes as the filters above discard all updates for p1. Thus, *GILL* retains all updates from VP2 by applying the filters depicted in Fig. 5b. Now, *GILL* collects four updates, the same number as in the current approach with only two VPs but no filters (Fig. 5a). But these four updates allow all three objectives—detecting the failure on 2—4, the hijack on p3, and identifying the origin AS of every prefix.

In practice, it is challenging to find from which VPs *GILL* should retain all updates as there is no ground truth. Selecting them randomly is an obvious option that performs poorly as *GILL* would retain all updates from many redundant VPs (Fig. 6) and discard updates from more valuable VPs. Thus, *GILL* uses algorithms that quantify redundancy between every pair of VPs and select a set of VPs that minimizes overall redundancy among the collected routes (§6). *GILL* keeps the full RIBs and all updates from these valuable (or *anchor*) VPs and filters updates received from other VPs.

6 *GILL*'S SAMPLING ALGORITHMS

We overview the two sampling algorithms that *GILL* uses to find redundant updates and anchor VPs, respectively. For reproducibility, we provide formalization, examples, and describe parameter calibration in §17-§18.

Component #1: Finding redundant BGP updates. *GILL* computes redundancy between past collected updates using the following three-steps algorithm.

Step 1 (§17.1): *GILL* takes a past set of updates β collected during a two-day period and groups updates that appear together in a short time window of 100s into *correlation groups*. These *correlation groups* capture groups of correlated updates, i.e., that often appear together. *GILL* builds *correlation groups* on a per-prefix basis, i.e., two updates with different prefixes cannot be in the same *correlation group*. In Fig. 5, the two updates for p1 collected by VP1 and VP2 are in the same *correlation group* whereas the two updates for p2 are in another *correlation group*. *GILL* then weights every *correlation group* based on how many times its updates appear together during the time window where the set of updates β was collected.

Step 2 (§17.2): GILL identifies redundant updates individually for every prefix using a metric called the *reconstitution power*.
Reconstitution power intuition: If we can identically reconstitute the set of updates β from one of its subsets α, then α contains useful updates and $\beta \setminus \alpha$ contains redundant updates. Two identical updates come from the same VP and have the same prefix, AS path, community values, and timestamp[1]. GILL's updates reconstitution algorithm works as follows: for every update u in α, GILL reconstitutes all the updates in the *correlation group* with the highest weight among the ones that include u. The *reconstitution power* is the percentage of updates in β that GILL can identically reconstitute.

For each prefix, GILL builds the set α of nonredundant updates by iteratively adding to α the set of all updates collected by the same VP and that most improves the *reconstitution power*. Observe that GILL adds to α either all updates collected by a VP, or none of them, as the filters that GILL builds match on the prefix and the VP only and cannot discriminate updates based on their AS path or community values (§7). GILL stops adding new updates to α when it can reconstitute 94% of the updates in β, which we experimentally find to be the best tradeoff between volume of data retained and loss of nonredundant information. With RIS and RV data, GILL stops when $|\alpha|/|\beta| \approx 0.16$, i.e., GILL can reconstitute 94% of a set of updates from only ≈16% of them. GILL classifies updates in α (16%) as nonredundant and the ones in $\beta \setminus \alpha$ (84%) as redundant.
Step 3 (§17.3): Finally, GILL exploits redundancy across prefixes as we find that two prefixes can be subject to similar and simultaneous route updates (e.g., when these two prefixes are announced by the same AS, which is the case for p1 and p2 in Fig. 5). If different prefixes are subject to the same updates that GILL classified as nonredundant in *Step 2*, then GILL classifies the updates for one of these prefixes as nonredundant and the others as redundant. Now, GILL classifies 93% of the RIS and RV updates as redundant ($|\alpha|/|\beta| \approx 0.07$).

Component #2: Finding anchor (i.e., valuable) VPs. GILL identifies anchor VPs (from which it retains all updates) by computing redundancy between combinations of updates collected by each VP, i.e., it quantifies how similar the views of the VPs are, using the following four steps.
Step 1 (§18.1): GILL selects a large, unbiased set of BGP events to gauge pairwise redundancy between VPs. First, GILL carefully selects three types of *non-global* events (path changes, outages, and origin changes). GILL avoids global events since all VPs tend to see them, rendering them less discriminating. Second, GILL stratifies its sample of events across space and time to avoid bias.
Step 2 (§18.2): GILL characterizes how VPs experience selected events. That is, for every BGP event, GILL computes the difference induced by the event on the topological features [58] of the ASes involved, as observed by each VP. These features embed information about the four attributes of a BGP update: time, prefix, AS path, and community values.
Step 3 (§18.3): GILL computes pairwise redundancy scores between VPs. That is, for every event, GILL computes the pairwise Euclidean distance in a n-dimensional space, where n is the number of topological features. VP pairs with similar feature values for an event are close in this space and thus likely redundant. GILL then computes the average Euclidean distance between each pair of VPs over all the selected events.
Step 4 (§18.4): GILL selects a set of anchor VPs, considering each VP's data *redundancy* and its *volume*. GILL considers the volume of data generated by every VP to prioritize the one that provides richer information within a few updates. GILL first selects the VP with the lowest average Euclidean distance to all other VPs, then greedily adds the VP that balances maximal Euclidean distance to already selected VPs and minimal additional data volume that the VP brings. GILL stops selecting new VPs when each remaining VP has the highest possible redundancy score with one selected VP. Intuitively, the higher the VP coverage, the lower the proportion of selected anchor VPs. With RIS and RV, GILL finds 178 anchor VPs.

[1]We use a 100s slack when comparing timestamps.

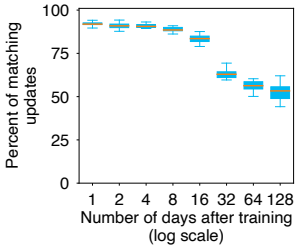


Figure 7: Ability of the generated filters to discard updates over time.

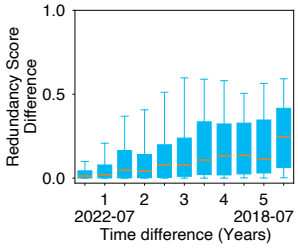


Figure 8: Redundancy score differences between two runs of *GILL*.

7 GILL'S FILTER GENERATION

Once GILL identifies redundant BGP updates, it computes filters to apply to its peering sessions.
Filtering policy. GILL builds filters that aim to discard redundant BGP updates that do not come from an anchor VP, and retain all others. We infer from experimental analyses the frequency at which GILL must refresh its filters. Fig. 5b gives an example of the filters generated by GILL. The first filter accepts all updates from anchor VP2. The subsequent (lower priority) filters discard redundant updates from other VPs. GILL employs an "accept everything" default filtering policy. Thus, GILL always retains new updates (i.e., not seen before), which ensures retention of updates from newly deployed VPs. GILL might discard new updates only when it relaunches its sampling algorithms and refreshes its filters (see next paragraph).
Keeping filters accurate over time. GILL needs an up-to-date list of redundant updates, otherwise an increasing number of new redundant updates match none of the filters over time and thus GILL retains them (due to the "accept everything" default policy). We infer the frequency at which GILL must refresh its filters (i.e., execute components #1 and #2 in §6) from experimental analyses. GILL *executes component #1 every 16 days.* We evaluate how accurate GILL's redundant update inferences remain over time. More precisely, we build GILL's filters using data from Sept. 1, 2023, and measure their ability to discard redundant updates in a set of updates collected d days after the filter generation, with d ranging from 1 to 128 days. Fig. 7 shows the percentage of updates matched

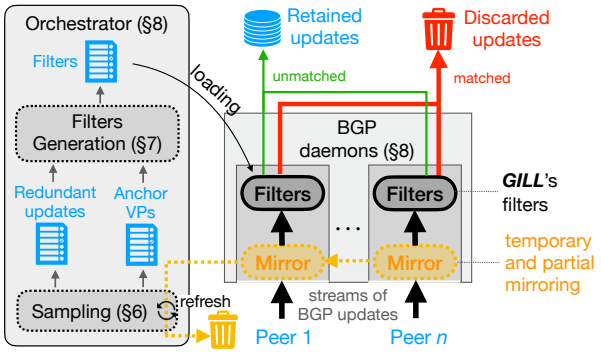


Figure 9: *GILL*'s workflow. All blue items are publicly available at https://bgproutes.io (§9).

Number of peers		100	1000	10000
With filters (i.e., GILL)				
Update load (per hour)	Average (28K upd/h)	0%	0%	0%
	99th percentile (241K upd/h)	0%	0%	high
Without filters				
Update load (per hour)	Average (28K upd/h)	0%	0%	39%
	99th percentile (241K upd/h)	0%	32%	high

Table 1: Proportion of updates lost by our BGP daemons as a function of the update frequency when using only one CPU. A green cell means daemons cope with the update frequency (no updates are lost) whereas a red cell means daemons drop at least one update. When the number of lost updates cannot be precisely computed because the load is too high, we just label it as high and color it in red.

by the filters (thus discarded by *GILL*). Logically, the higher the value of d, the lower the percentage of matched updates, as the proportion of new updates (i.e., never observed before) in a set of updates increases when d increases. We configure *GILL* to execute component #1 in §6 every 16 days, as it appears to be the threshold after which the percentage of matched updates critically drops.

GILL *executes component #2 every year.* To estimate how often *GILL* should refresh its list of anchor VPs, we compute the redundancy score for every pair of VPs on Sept. 2023 and compare them with redundancy scores computed m months before. Fig. 8 shows the distribution of the redundancy score differences for different values of m, ranging from six to 66 (=5.5 years). Logically, the higher the value of m, the higher the redundancy score differences. However, when $m \leq 12$, the redundancy score differences are low (the difference is below 0.1 in the median case), and we find that redundancy scores change by less than 5%. We thus configure *GILL* to execute component #2 in §6 one time per year.

Observe that *GILL*'s operators can temporarily overwrite these default settings to accommodate bursts of new peering sessions, e.g., when the platform bootstraps.

Discarding future redundant updates. One challenge when generating filters is that *GILL* identifies *past* redundant updates whereas filters aim to discard *future* redundant updates, which are impossible to predict. Filters that match on all attributes (i.e., prefix, AS path, community values) and the sending VP would be too fine-grained and result in continuous retention of new updates. In Fig. 5, new updates induced by the failure have an AS path likely never observed before. Fine-grained filters that match on the AS path would retain all these updates, increasing the volume and redundancy of data collected.

However, updates classified as redundant by *GILL* in §6 at time $t1$ are often similar to updates classified as redundant at time $t2$ ($t1 < t2$). *GILL* thus generates coarse-grained filters that match only on the VP from which it receives an update and its prefix (Fig. 5b). Such a filter thus matches on an entire space of similar (and redundant) updates—which are either all retained or all discarded. We confirm the validity of this approach by developing two modified versions of *GILL* that build finer-grained filters. The first version (*GILL -asp*) builds filters that match on the prefix, VP, and AS path. The second version (*GILL -asp-comm*) builds filters that also match on community values. We then use a typical training-testing pipeline to evaluate the proportion of future redundant updates that the generated filters discard. More precisely, we consider the set of redundant updates r ($= \beta \setminus \alpha$) inferred by *GILL*, which we divide into two distinct sets $r1$ and $r2$ that are consecutive in time (i.e., updates in $r2$ appear after the ones in $r1$). We then generate filters with the three versions of *GILL* that match on the updates in $r1$ and measure the proportion of updates in $r2$ that match the filters. We find that *GILL*'s filters match 87% of the updates in $r2$ against only 43% for *GILL -asp* and 0% for *GILL -asp-comm*.

Observe that *GILL* generates filters that match on updates inferred as redundant by its sampling algorithms, which we purposely designed to align with the coarse-grained granularity of filters: They classify either all or no updates for a given prefix and VP as redundant (§6). Thus, filters cannot match an update inferred as nonredundant by *GILL*.

8 SOFTWARE

Fig. 9 describes the overall workflow of *GILL*, which relies on the following two software components:

A custom BGP daemon, written in C and tailored to peer with a single BGP router, apply filters on received updates, and store (either RIBs every eight hours or every update) updates not matched by the filters. We evaluate the ability of our BGP daemon to cope with high data volume by running multiple instances of it simultaneously on a single 3.20 GHz Apple M1 Pro CPU with 16GB of RAM. For every BGP daemon that we run, we configure a fake peer that establishes a BGP session with the daemon and sends a stream of BGP updates. Table 1 shows the percentage of BGP updates lost by the BGP daemons as a function of the BGP update frequency and depending on whether they apply filters generated by *GILL* (top part) or not (bottom part). We configure our fake peers to send BGP updates at a frequency that is either the average (28k/hour) or the 99th percentile (241k/hour) of the frequency at which RIS and RV peers send updates.

We find that a single CPU successfully handles (i.e., losing no updates) up to 10k BGP daemons with the average update frequency

and up to 1k daemons with the high update frequency (99th percentile) when using filters. Thus, we expect *GILL* to support tens of thousands of BGP sessions (even during peak times) on a server with many CPUs. Logically, we observe that our BGP daemons can process more updates when using filters because less data is written to disk, which is the most time-consuming task of our daemon.

Observe that *GILL* could implement filters using *route-maps* as suggested in [4]. However, we find that a server with 24 CPUs and 64GB of RAM running FRR [45] (a typical software router that current collection platforms use to peer with VPs) can only handle up to $\approx 10K$ route-maps, far fewer than what *GILL* generates (≈1M). *An orchestrator,* written in Python, starts new BGP sessions using our daemon, periodically executes *components #1* and *#2* of *GILL*'s sampling algorithms (§6), generates filters (§7), and loads them into the BGP daemons. Observe that sampling algorithms require *all* data (all updates from all VPs), which conflicts with the *overshoot-and-discard* principles of *GILL*. However, *steps #1* and *#2* of *component #1* can execute prefix by prefix and *step #3* only needs the output of the previous two steps. Additionally, *component #2* can run on short windows of updates. Thus, the orchestrator copes with high data volume by temporarily retaining all data for a prefix or all data during a short time window (using a mirroring scheme invisible to users, see Fig. 9), executing *components #1* and *#2*, generating filters, and then dropping this data. Finding redundant updates (*component* #1) takes 22 hours while finding anchor VPs (*component* #2) takes 35 hours with the RIS and RV data. These execution times are compatible with the frequency at which *GILL* updates its filters (§7).

9 *GILL* IS UP AND RUNNING

GILL runs at https://bgproutes.io and currently collects BGP updates from a few routers. The installation of new VPs is automated: operators can connect their BGP routers to *GILL* by submitting a form on the website. *GILL* automatically configures new peering sessions based on the information provided in the form and new peers are visible on the website within a few minutes. *GILL* minimizes the risk of fake or misconfigured peering sessions using a two-step authentification scheme: *(i)* a new participant must send an email to *GILL* with the AS number provided in the form *(ii)* once received, *GILL* cross-checks that the email address of the sender owns that AS according to PeeringDB [43]. In addition to its own peers, *GILL* also takes as input streams of BGP updates from all RIS VPs using the WebSocket API of RIS Live [48] and all RV VPs using a custom proxy that gathers and gives to *GILL* the RV data in near real-time. Overall, *GILL* currently processes and stores data for ≈2500 VPs. *GILL* stores the collected BGP updates in a public database using the MRT format [8] with *Bzip2* file compression. We publish two supporting documents:

- The computed filters from which users can infer which BGP updates are discarded by *GILL* and possible missing in the database;
- The list of anchor VPs found by *GILL* and from which all the updates are processes and stored.

These documents help users find which bits of data they should process from RIS when they have limited resources, which mitigates the risk of common but naive sampling approaches (§16).

10 BENCHMARKING *GILL*'S SAMPLING

We show that *GILL*'s sampling improves the trade-off between data volume and information inferred compared to current BGP data sampling schemes in five use cases.

Use cases. We carefully picked the five use cases such that each BGP attribute is required for at least one of them. For instance, the *time* is required to detect transient events (use case *I*); the *prefix* is required to detect Multiple Origin ASes (MOAS) prefixes (use case *II*); the *AS path* is required to map the Internet topology (use case *III*); and the *community values* are required to detect action communities (use case *IV*) and unchanged-path updates (use case *V*). These use cases allow us to show that *GILL*'s sampling does not overfit on some particular use cases or BGP attributes. For each use case, we process updates collected by all RIS and RV VPs during 30 one-hour periods (randomly selected in Sept. 2023), and benchmark *GILL*'s sampling on a set of events found. We briefly describe below each use case along with our experimental settings.

I Transient paths detection. Transient paths are BGP routes visible for less than five minutes, a typical BGP convergence delay [30], and which can be attributed to e.g., path exploration [39]. We focus on all transient path events detected during the 30 hours, a total of 859K events.

II MOAS prefixes detection. MOAS prefixes are announced by multiple distinct ASes [56], due to legitimate [66] or malicious [15, 51, 59] actions. We use the methodology of [46] to eliminate false positives. We focus on all 1587 MOAS observed during the 30 hours.

III AS topology mapping. This is useful for e.g., inferring BGP policies [31] or AS paths [33]. For each VP, we process the first RIB dump of Sept. 2023 as well as the updates collected during the 30 one-hour periods. We focus on all 687K distinct AS links observed.

IV Action communities detection. Action communities are associated with traffic engineering actions and are the most challenging to observe [60]. We consider all 8683 action communities provided in [60] and observed during the 30 hours.

V Unchanged-path updates detection. Unchanged-path BGP updates are announcements that only signal a change in community values but not in AS path [29]. We consider all 263K unchanged-path updates observed during the 30 hours.

Baselines. We benchmarked *GILL*'s sampling against several baselines from the following four categories.

GILL-simplified: We developed two simple versions of *GILL*, one named *GILL-upd* that samples at the update granularity (using *Component #1* in §6) and another named *GILL-vp* that samples at the VP granularity (using *Component #2* in §6).

Naive baselines: We develop four naive sampling schemes some of which are used in practice (see §16): *(i) Rnd.-VP* selects updates exported by a random set of VPs *(ii) AS-Dist.* selects a first VP randomly and the next ones to maximize the AS-level distance between selected VPs. It collects all updates from them; *(iii) Unbiased* takes all VPs, iteratively removes the one that most increases the bias (according to the definition in [57]), and collects all updates from the remaining VPs, and *(iv) Rnd.-Upd* selects updates randomly regardless of which VP they come from.

Definition-based specifics: We compare *GILL* against three *specific* sampling schemes optimized for minimizing redundancy based on redundancy definitions 1, 2, and 3 in §4.

	Sampling Scheme	*GILL*	*GILL*-simplified		Naive				Definition-based specifics			Use-case-based specifics				
			GILL-upd	*GILL*-vp	Rnd. Upd.	Rnd. VP	AS-Dist.	Unbiased	Def. 1	Def. 2	Def. 3	*I*	*II*	*III*	*IV*	*V*
Use cases	**Trans. path detection (*I*)**	96%	65%	75%	83%	75%	95%	89%	76%	98%	82%	100%	83%	79%	82%	83%
	MOAS detection (*II*)	95%	95%	45%	33%	35%	59%	46%	47%	48%	40%	36%	100%	32%	33%	32%
	Topo mapping (*III*)	90%	93%	61%	72%	41%	67%	38%	43%	49%	33%	72%	72%	100%	72%	64%
	Action Coms. detection (*IV*)	91%	95%	49%	79%	48%	41%	42%	46%	45%	44%	78%	77%	85%	100%	73%
	Unchanged-path Upd. detection (*V*)	87%	60%	80%	76%	74%	43%	61%	63%	63%	90%	76%	76%	71%	76%	100%

Table 2: *GILL*'s sampling outperforms all naive baselines, for all use cases. Unlike the *Use-case-based specifics* baselines, *GILL* avoids overfitting. The color means that *GILL* performs better ( ), worse ( ), or similarly ( ) than the baseline.

Use-case-based specifics: We compare *GILL* against five *specific* sampling schemes, one optimized for each of the five use cases described above. These *specific* sampling schemes *optimize the trade-off between the volume of the data and its capacity to achieve a particular objective*. For instance, the specific sampling scheme optimized to map the AS topology (use case *III*) iteratively selects the VP that best improves the trade-off between the number of discovered AS links and the volume of processed data.

Benchmark results. We compute for *GILL* and each baseline the proportion of events that they detect or links that they observe and report the results in Table 2. For instance, *GILL* detects 95% of MOAS events means that the data *GILL* samples enables to detect 95% of the 1587 MOAS events used in the benchmark. The cell of a baseline is green when *GILL* outperforms the baseline, red if the baseline is better, and yellow if the two perform similarly (±5%). We ensure that the baselines process the same number of updates as *GILL*, i.e., 6.7% of RIS and RV updates (see §6).

Takeaway #1: Unlike its simplified versions, *GILL* performs well for every use case. *GILL-upd* performs poorly for use cases *I* and *V* whereas *GILL-vp* always performs worse than *GILL*. *GILL-vp* outperforms *GILL-upd* for use cases *I* and *V* likely because the higher link visibility that *GILL-upd* enables compared to *GILL-vp* is not helpful for these use cases. *GILL-upd* outperforms *GILL-vp* for other use cases, as it collects more diverse BGP attributes. This complementarity between *GILL-vp* and *GILL-upd* demonstrates that *Ingredient #2* in *GILL*'s principles (§5) is sound.

Takeaway #2: *GILL* outperforms each naive baseline for every use case, and sometimes significantly, e.g., *GILL* detects +62%, +60%, +36%, and +49% MOAS hijacks (use case *II*) compared to *Rnd.-Upd.*, *Rnd.-VPs*, *Dist.-based* and *Unbiased*, respectively.

Takeaway #3: The *definition-based specifics* perform poorly for many use cases, e.g., *GILL* detects +45%, +46%, and +47% action communities (use case *V*) compared to the *specific* optimized for Def. 1, 2, and 3, respectively. This demonstrates that *GILL*'s filter generation is sound (§7).

Takeaway #4: *GILL* generalizes whereas *use-case-based specific* sampling schemes overfit. In fact, a *specific* baseline optimized for a use case outperforms *GILL* for that use case (thus the diagonal is yellow/red). However, *GILL* always outperforms a *specific* baseline for the use cases this *specific* baseline does not optimize. These results demonstrate that our algorithms in §6 avoid overfitting.

11 LONG-TERM IMPACT

The long-term impact of *GILL* will only be visible when it will peer with thousands of BGP routers. Yet, we can evaluate the long-term impact now using simulations with C-BGP [47].

Used AS topologies and settings. We generate a *pruned known AS topology* and an *artificial topology* using the methodology in §3 except that they now all have 1k ASes to limit the computational resources needed.

Use cases. We use the three use cases in §3 but focus on p2p links for both *topology mapping* and *failure localization* as they are the more challenging to capture, and on Type-1 forged-origin hijacks for the *hijack detection* use case as they are the most common [25].

Baselines. We compare *GILL*'s sampling against two baselines.

Random VPs: We iteratively select a VP among the deployed VPs and collect the updates that it exports until the total number of collected updates has reached the number of updates retained by *GILL*. We use this baseline as it is commonly used in practice according to our survey (§16).

Best case: Akin to our simulations in §3, we take all updates from all deployed VPs, which is a best-case scenario. Inevitably though, *best case* processes more updates than *GILL*.

Simulations settings. We tested different coverage (i.e., number of ASes that deploy a VP) ranging from 2% (the rounded-up coverage of RIS and RV) to 100% (all ASes host a VP). Observe that *GILL*'s sampling algorithms (§6) are data-driven and thus need past BGP updates as input. We thus generate 500 random link failures (distinct from the failures used for the *failure localization* use case) and feed *GILL* the induced BGP updates collected by every deployed VP.

Simulation results. Unlike in §10 where we focus on a set of events observed in the RIS and RV data, we now have the ground truth. We thus compute, for each sampling scheme, the proportion of events that they detect among *all* the events that we triggered or links that they observe among *all* the links that exist (Table 3). Note that Table 3 shows results with the *artificial topology*, results with the *pruned known AS topology* are similar.

Takeaway #1: *GILL* responds to the increasing coverage by discarding more data. *GILL* retains 18% of the updates when coverage is 2%, and 7.9%, 5.4%, 4.7%, and 4.4%, when coverage is 10%, 20%, 50%, and 100%, respectively. Similarly, *GILL* finds that 17% of the VPs are anchors when coverage is 2%, and 3.3%, 1.3%, 0.9%, and 0.4% when coverage is 10%, 20%, 50%, and 100%, respectively. This behavior is expected as the higher the coverage the higher the proportion of redundant updates.

Coverage		2%			10%			25%			50%			100%		
Data Collection Scheme		GILL	Rnd. VP	Best Case	GILL	Rnd. VP	Best Case	GILL	Rnd. VP	Best Case	GILL	Rnd. VP	Best Case	GILL	Rnd. VP	Best Case
Updates retained / Anchor VPs		*18.0% / 17.0%*		100%/100%	*7.9% / 3.3%*		100%/100%	*5.4% / 1.3%*		100%/100%	*4.7% / 0.9%*		100%/100%	*4.4% / 0.4%*		100%/100%
Use cases	Topology Mapping	14%	4%	20%	33%	7%	50%	42%	7%	69%	61%	16%	85%	78%	25%	100%
	Failure Localisation	29%	11%	37%	61%	14%	81%	60%	25%	80%	80%	18%	92%	94%	40%	100%
	Hijack detection	58%	53%	73%	73%	54%	87%	77%	59%	92%	82%	74%	96%	85%	76%	100%

Table 3: Performance of *GILL*, *Rnd.-VPs* and *best-case* on a simulated mini Internet where the proportion of ASes deploying a VP ranges from 1% to 100%. *GILL* leverages high coverage but the two other baselines do not.

Takeaway #2: *GILL*'s *overshoot-and-discard* data collection scheme is efficient. While *best-case* outperforms *GILL*, it also collects many more updates. With 50% coverage, *GILL* localizes 80% of p2p links against 92% with *best-case*. However, *GILL* collects ≈21× fewer updates than *best-case*. When coverage is high (e.g., 50%), the number of updates processed with *GILL* is comparable to *best-case* with a 2% coverage. Assuming this observation holds in the real Internet, *GILL* would collect a similar number of updates as RIS and RV today while peering with 50% of the ≈75k ASes, which would triple the number of p2p links mapped, double the number of localized failures, and reduce by 33% the proportion of undetected hijacks.
Takeaway #3: *GILL* outperforms *random VPs* for all use cases. Even with 100% coverage, only 25% of p2p links are detected when processing the same number of updates as *GILL*. The forged-origin hijack use case is the more challenging for *GILL*, as all prefixes owned by an AS are subject to identical updates in our simulations (thus *GILL* discards many of these updates) but only one prefix is hijacked. Yet, *GILL* always outperforms *random VPs* for this use case, e.g., *GILL* detects +18% of hijacks with a 25% coverage.

12 IMMEDIATE BENEFITS

We show that running *GILL*'s sampling algorithms on RIS and RV data improves coverage and accuracy of three studies/tools. Unlike in §11, the ground truth is now unknown.

GILL helps to infer +16% more AS relationships. We replicate the methodology proposed in [31] that uses BGP data from RIS and RV to infer AS relationships and build the widely-used CAIDA AS-relationship dataset [19]. We compute the number of inferred AS relationships for every month in 2023 when using the 648 VPs that CAIDA uses to build its dataset (in Jan. 2023) and when using all the RIS and RV data but sampled using *GILL*'s algorithms. We ensure that *GILL* retains the same number of BGP updates as the data set CAIDA collected from the 648 VPs. Thus, we can attribute any performance improvement to *GILL*. We find that *GILL* collects updates that enable consistent (from Jan. 2023 to Dec. 2023) inference of ≈89k additional AS relationships (≈+17%) while missing only ≈8k AS relationships (≈1.5%) present in the original dataset. We also replicated the AS relationship validation algorithm used in [31] (which relies on IRR and RIR data) and found that the true positive rate (the metric used in [31]) remains identical (97%). We conclude that *GILL* enables inference of ≈+16% more AS relationships compared to the original dataset provided by CAIDA, while processing the same number of BGP updates and without losing accuracy.

GILL reduces flawed inferences in the ASRank dataset. We replicate the methodology used by ASRank [11], which uses 648 VPs to compute the AS Customer Cone Sizes (CCS). We find that the CCS changes for 1067 ASes when using the same number of BGP updates as in [11] but sampled using *GILL*. We manually investigated a few cases of substantial changes and found that inferences made using *GILL* are more accurate. For instance, AS132337 has an incorrect (confirmed by AS132337 itself) CCS of 1 in the original dataset and a correct CSS of 18k when using *GILL*. Similarly, AS24745 is the route server of Balcan-IX and has an incorrect CSS of 16 in the original ASrank dataset, which is fixed when using *GILL* (CSS is one). We observe that *GILL* enables more accurate inferences of CCSs because it collects more diverse AS paths.

GILL improves forged-origin hijack inferences. We replicate the algorithm of DFOH [25] that uses routes collected by 287 RIS and RV VPs to infer forged-origin hijacks in September 2023. We implement two versions of DFOH, one called DFOH_{GILL} which uses BGP routes collected by *GILL*, and another one called DFOH_{R} that uses routes collected from a random set of VPs. In both versions, we ensure that the number of routes collected is identical to the one used in [25]. As DFOH relies on probabilistic inference, we measure the performance of DFOH_{GILL} and DFOH_{R} in terms of True Positive Rate (TPR) and False Positive Rate (FPR). We obtain an approximation of ground truth (needed to compute the TPR and FPR) by implementing a third version of DFOH, called DFOH_{ALL} that uses all the RIS and RV data. Note that DFOH_{ALL} is an approximation of ground truth as incorrect inferences are still possible even if all the data is used because of the low RIS and RV coverage. DFOH_{GILL} uncovers 1708 suspicious cases against only 1300 for DFOH_{R}. DFOH_{GILL} outperforms DFOH_{R} for both the TPR and the FPR: It has a TPR of 94% (against 71.5% for DFOH_{R}) and a FPR of 14.4% (against 60.1% for DFOH_{R})—a ≈4× better precision.

13 RELATED WORK

Existing BGP routes collection platforms. Public BGP route collection systems include RIS (≈1500 VPs) [49], RV (≈1000 VPs) [61], PCH (≈700 VPs) [42], BGPWatch (15 VPs) [7] and Isolario (not maintained anymore) [27]. Private collection systems include bgp.tools (≈1000 VPs) [17], PacketVis (≈2000 VPs) [41], Radar by QRator (≈800 VPs), Kentik's and ThousandsEyes's BGP route monitoring platforms (confidential number of VPs). Observe that their coverage (when known) is always tiny (<2%). However, they could all benefit from *GILL*'s algorithms to increase their coverage with limited cost.

VPs deployment schemes. Current VPs deployment schemes suggest deploying a few but strategically positioned VPs [53]. For

instance, Gregori et al. proposed a methodology that finds a relevant placement for a new VP [24], and Cittadini et al. demonstrated the marginal utility of adding new VPs at the core of the Internet [16]. The peering strategy of RIS is to maximize coverage on the core of Internet (e.g., by peering with Tier1 ASes) and improve coverage diversity across countries [2]. *GILL*'s approach is radically different: deploying many VPs but discarding redundant routes.

BGP data sampling schemes. Prior works suggest sampling BGP data at the VP granularity. For instance, Zhang et al. and Oliviera et al. show that carefully selecting VPs increases the utility of the data [38, 65]. But their technique is tailored for topology mapping whereas *GILL*'s algorithms are not specific to a particular objective.

14 FUTURE DIRECTIONS

If *GILL* gains traction, we expect it to trigger new interesting research problems and future directions.

Incentivizing network operators to peer with GILL. Our vision for *GILL* includes an order of magnitude increase in the number of peers, which motivates the question: how do we inspire such an expansion in participation? We have already taken two steps to improve the cost-benefit calculus of peering with *GILL*: a fully automated and immediate peering session activation via a web form (§9); and a bootstrap of *GILL* with the 2500 peering sessions from RIS and RV to ensure a head start in visibility (§9).

Two other strategies could further incentivize participation.

- **Custom services that improve visibility.** In return for peering, *GILL* could let the network operator configure forwarding rules such that *GILL* forwards some updates to the operator's network prior to discarding them. Forwarding rules would typically enable operators to have high visibility of their prefixes. If *GILL* had 100% coverage of VPs, operators could make hijack detection systems such as ARTEMIS [56] bulletproof for their prefixes.
- **Collective action.** Recent community-driven routing security initiatives such as MANRS [32] or VIPzone [18] could encourage participants to contribute BGP data to public BGP data collection platforms. The FCC's recent notice of proposed regulation [63] to require disclosure of BGP security strategies could lend further motivation to such strategies.

Preventing fake peering sessions and data. While *GILL* includes a basic authentification scheme when installing a new peering session, nothing prevents an attacker with an AS from announcing fake updates once it peers with *GILL*. Remote peering sessions also enable on-path attackers to modify the content of the BGP messages to replace route updates with fake ones.

Fake BGP updates and on-path attackers are also possible with current collection platforms, which as far as we know, do not employ any mechanism that consistently verifies the validity of the collected routes. Thus, *GILL* opens up new research problems in verifying the correctness of the collected BGP updates. Encrypted BGP peering sessions using e.g., BGP over QUIC seems a promising starting point [13, 64].

Generalizing to other types of Internet routing data. The principles used in *GILL*'s algorithms and implementation extend to other types of BGP monitoring systems (e.g., BMP) and potentially other types of Internet data, e.g., active measurement platforms (e.g., RIPE Atlas [50]). Adapting our algorithms for these use cases is a promising direction.

Ethics. See §16 about the ethics of our survey. Otherwise, this work does not raise any ethical issues.

ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers and our shepherd Marinho Barcellos for their feedback. We thank Ben Cox (bgp.tools [17]) and Hans Kuhn (RouteViews [61]) for their feedback at the early stage of this work. This work was supported by the ArtIC project (grant ANR-20-THIA-0006-01), Région Grand Est, Inria Nancy-Grand Est, IHU of Strasbourg, U. of Strasbourg, U. of Haute-Alsace, the RIPE Community Fund Project, Silicon Valley Foundation for Cisco (CG1318167 and CG1348196), NSF CNS-2120399, NSF OAC-2131987, and NSF OAC-2029309. The views and conclusions are those of the authors and do not represent the policies or endorsements of the funding agencies.

REFERENCES

[1] Emile Aben. 2020. Route Collection at the RIPE NCC - Where are we and where should we go? https://labs.ripe.net/author/emileaben/.
[2] Emile Aben. 2023. Two Years of Selective Peering with RIS. https://labs.ripe.net/author/emileaben/two-years-of-selective-peering-with-ris/.
[3] Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. 2015. Hyperbolic graph generator. In *Computer Physics Communications*.
[4] Thomas Alfroy, Thomas Holterbach, Thomas Krenc, KC Claffy, and Cristel Pelsser. 2023. Internet Science Moonshot: Expanding BGP Data Horizons. In *HotNets '23*. ACM.
[5] Lorenzo Ariemma, Mariano Scazzariello, and Tommaso Caiazzi. 2021. MRT#: a Fast Multi-Threaded MRT Parser. In *IFIP/IEEE IM '21*.
[6] BGPKIT. 2022. BGPKIT. https://blog.bgpkit.com/.
[7] BGPWatch. 2023. BGPWatch: BGP Routing Analysis and Diagnostic Platform. https://bgpwatch.cgtf.net/.
[8] Larry Blunk, Craig Labovitz, and Manish Karir. 2011. Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format. In *RFC 6396*.
[9] Paolo Boldi and Sebastiano Vigna. 2013. Axioms for Centrality.
[10] Timm Böttger, Félix Cuadrado, and Steve Uhlig. 2018. Looking for hypergiants in PeeringDB. In *SIGCOMM CCR*.
[11] CAIDA. 2023. AS Rank. https://asrank.caida.org/.
[12] Nikolaos Chatzis and al. 2013. On the benefits of using a large IXP as an internet vantage point. In *IMC*. ACM.
[13] Shuanglong Chen, Yongkang Zhang, Haibo Wang, and Zhenbin Li. 2021. *BGP Over QUIC*. Technical Report draft-chen-idr-bgp-over-quic-00.
[14] CIDR. 2023. CIDR REPORT. https://www.cidr-report.org/as2.0/.
[15] CitizenLab. 2012. A Case Study of the China Telecom Incident. https://citizenlab.ca/2012/12/.
[16] Luca Cittadini, Stefano Vissicchio, and Benoit Donnet. 2014. On the quality of BGP route collectors for iBGP policy inference. In *IFIP '14*.
[17] Ben Cox. 2023. BGP tools. https://bgp.tools/.
[18] David Clark and Cecilia Testart and Matthew Luckie3 and kc claffy. 2024. A path forward: Improving Internet routing security by enabling zones of trust. *Journal of Cybersecurity* (2024).
[19] University San Diego. 2022. The CAIDA AS Relationships Dataset, 2022. https://www.caida.org/catalog/datasets/as-relationships/.
[20] Benoit Donnet and Olivier Bonaventure. 2008. On BGP Communities. In *SIGCOMM CCR*.
[21] Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs. 2004. Locating Internet Routing Instabilities. *ACM SIGCOMM* (2004), 205–218.
[22] Linton C. Freeman. 1978. Centrality in social networks conceptual clarification. *Social Networks* (1978).
[23] Lixin Gao and Jennifer Rexford. 2000. Stable Internet Routing without Global Coordination. In *SIGMETRICS '00*.
[24] Enrico Gregori, Alessandro Improta, Luciano Lenzini, Lorenzo Rossi, and Luca Sani. 2012. On the Incompleteness of the AS-Level Graph: A Novel Methodology for BGP Route Collector Placement. In *IMC '12*.
[25] Thomas Holterbach, Thomas Alfroy, Amreesh Phokeer, Alberto Dainotti, and Cristel Pelsser. 2023. A System to Detect Forged-Origin BGP Hijacks. In *NSDI'24*.
[26] Thomas Holterbach, Stefano Vissicchio, Alberto Dainotti, and Laurent Vanbever. 2017. SWIFT: Predictive Fast Reroute. In *ACM SIGCOMM*.

[27] Alessandro Improta. 2023. Isolario project: The real-time Internet routing observatory. https://content.cooperate.com/post/internet_history/.
[28] Robert Kisteleki. 2023. RIPE NCC Measurement Data Retention Principles. https://labs.ripe.net/author/kistel/ripe-ncc-measurement-data-retention-principles/.
[29] Thomas Krenc, Robert Beverly, and Georgios Smaragdakis. 2020. Keep Your Communities Clean: Exploring the Routing Message Impact of BGP Communities. In *CoNEXT '20*.
[30] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. 2000. Delayed Internet Routing Convergence. In *SIGCOMM CCR*.
[31] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and kc claffy. 2013. AS Relationships, Customer Cones, and Validation. In *IMC '13*.
[32] MANRS. 2021. Mutually Agreed Norms for Routing Security. https://manrs.org/.
[33] Z. Morley Mao, Lili Qiu, Jia Wang, and Yin Zhang. 2005. On AS-Level Path Inference. In *SIGMETRICS '05*.
[34] Alexandros Milolidakis, Tobias Bühler, Kunyu Wang, Marco Chiesa, Laurent Vanbever, and Stefano Vissicchio. 2023. On the Effectiveness of BGP Hijackers That Evade Public Route Collectors. In *IEEE Access*.
[35] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. 2008. Path splicing. In *SIGCOMM '08*. ACM.
[36] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. 2006. Building an AS-Topology Model That Captures Route Diversity. In *SIGCOMM CCR*. ACM.
[37] University of Oregon. 2023. Route Views Peers list. http://www.routeviews.org/peers/peering-status.html.
[38] Ricardo Oliveira and al. 2006. Placing BGP monitors in the Internet. In *Technical No. UCLA, TR*.
[39] Ricardo Oliveira, Beichuan Zhang, Dan Pei, Rafit Izhak-Ratzin, and Lixia Zhang. 2006. Quantifying Path Exploration in the Internet. In *ACM IMC'06*.
[40] Chiara Orsini, Alistair King, Danilo Giordano, Vasileios Giotsas, and Alberto Dainotti. 2016. BGPStream: A Software Framework for Live and Historical BGP Data Analysis. In *IMC '16*.
[41] PacketVis. 2024. BGP and RPKI real-time monitoring. https://https://packetvis.com/.
[42] PCH. 2010. Packet Clearing House. https://www.pch.net/.
[43] PeeringDB. 2023. The Interconnection Database. https://www.peeringdb.com/.
[44] Lars Prehn and Anja Feldmann. 2021. How Biased is Our Validation (Data) for AS Relationships?. In *IMC '21*.
[45] FRRouting Project. 2023. A fully featured, high performance, free software IP routing suite. https://frrouting.org/.
[46] Lancheng Qin, Dan Li, Ruifeng Li, and Kang Wang. 2022. Themis: Accelerating the Detection of Route Origin Hijacking by Distinguishing Legitimate and Illegitimate MOAS. In *USENIX Security*.
[47] B. Quoitin and S. Uhlig. 2005. Modeling the routing of an autonomous system with C-BGP. *IEEE Network* (2005).
[48] RIPE. 1. RIPE RIS Live. https://ris-live.ripe.net/.
[49] RIPE. 1. RIPE RIS Raw Data. https://www.ripe.net/data-tools/stats/ris/.
[50] RIPE. 1. The RIPE Atlas measurement platform. https://atlas.ripe.net/.
[51] RIPE. 2018. YouTube Hijacking: A RIPE NCC RIS case study. http://www.ripe.net/internet-coordination/news/industry-developments/.
[52] RIPE. 2023. RIPE RIS Peers list. https://www.ris.ripe.net/peerlist/.
[53] Matthew Roughan, Simon Jonathan Tuke, and Olaf Maennel. 2008. Bigfoot, Sasquatch, the Yeti and other missing links: what we don't know about the AS graph. In *IMC '08*.
[54] Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertész. 2007. Generalizations of the clustering coefficient to weighted complex networks. *Phys. Rev. E* (2007).
[55] Sciencealert. 2018. Less Than 1% of Large Hadron Collider Data Ever Gets Looked at. https://www.sciencealert.com/over-99-percent-of-large-hadron-collider-particle-collision-data-is-lost.
[56] Pavlos Sermpezis and al. 2018. ARTEMIS: Neutralizing BGP hijacking within a minute. In *ToN*.
[57] Pavlos Sermpezis, Lars Prehn, Sofia Kostoglou, Marcel Flores, Athena Vakali, and Emile Aben. 2023. Bias in Internet Measurement Platforms. In *TMA'23*.
[58] Mattia Tantardini, Francesca Ieva, Lucia Tajoli, and Carlo Piccardi. 2019. Comparing methods for comparing networks. In *Nature*.
[59] Ars Technica. 2017. Russian-controlled telecom hijacks financial services' Internet traffic. https://arstechnica.com/security/2017/04/.
[60] Krenc Thomas, Luckie Matthew, Marder Alexander, and kc Claffy. 2023. Coarse-grained Inference of BGP Community Intent.. In *IMC'23*.
[61] Oregon Univ. 2021. Route Views Project. www.routeviews.org/.
[62] U.S. Federal Communications Commission. 2022. NOTICE OF INQUIRY. PS Docket No. 22-90. In the Matter of Secure Internet Routing.
[63] U.S. Federal Communications Commission. 2024. NOTICE OF PROPOSED RULEMAKING, In the Matter of Reporting on Border Gateway Protocol Risk Mitigation Progress and Secure Internet Routing. (2024). https://docs.fcc.gov/public/attachments/DOC-402609A1.pdf.
[64] Thomas Wirtgen, Nicolas Rybowski, Cristel Pelsser, and Olivier Bonaventure. 2023. Routing over QUIC: Bringing transport innovations to routing protocols.
[65] Ying Zhang, Zheng Zhang, Z. Morley Mao, Y. Charlie Hu, and Bruce M. Maggs. 2007. On the impact of route monitor selection. In *IMC '07*.
[66] Xiaoliang Zhao, Dan Pei, Lan Wang, Dan Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. 2001. An Analysis of BGP Multiple Origin AS (MOAS) Conflicts. In *IMW '01*.

15 APPENDICES

We provide appendices to support reproducibility and transparency of artifacts of this work. The appendices address questions of interest to a small minority of reviewers such as additional implementation detail on the algorithms in §6, including formalization, illustrative examples, and empirical grounding for selection of default threshold parameters. We will publish this material on *GILL*'s website. We expect reviewers to only use this material for reference purposes if at all. Appendices are supporting material that has not been peer-reviewed.

16 SURVEY

Detailed methdology. We selected eleven papers and classified them based on how authors collected the BGP data (categories C_1 and C_2, in Table 4). We then emailed the authors and asked them about their experience with using BGP routes from RIS and RV. We did not receive answers for three papers. Observe that we do not show parts of a few answers that would make de-anonymization possible. However, missing parts never change the message conveyed in the answers.

Detailed answers. Table 4 lists the questions we asked the participants of our survey along with their detailed answers. We color the answers based on our interpretation of whether the responses are aligned with *GILL*'s objectives (green) or not (red). Neutral answers are colored in blue. The vast majority of the answers indicate that *GILL* would be beneficial for users and improve the quality of their measurement studies.

Common BGP data sampling schemes. Among the seven respondents who took the data from a subset of the VPs, one picked geographically distant VPs. While intuitive, this strategy fails to optimize for some metrics (e.g., AS link coverage §10). Another respondent unintentionally removed some VPs (leaving an arbitrarily selected set in the study) and two did not remember how they selected their VPs. All the remaining respondent selected their VPs arbitrarily. We show in the benchmark (§10) that sampling data in an unoptimized fashion, i.e., arbitrarily or with simple metrics leads to poor performance for most of the use cases.

Ethics The participants of the survey freely participated. We contacted them by email to ask them whether they would agree to participate. We stated the purpose of the survey and notified them we might publish the results anonymously. Following is the exact wording we used when soliciting participants.

"I would like to know whether you would be willing to answer a quick survey about why you selected these VPs and the impact that you think this selection made on your measurement study.

Answering this survey will help us to better understand how researchers proceed when selecting BGP vantage points, why they often do not take them all, and what is the impact of the vantage points selection on the results of the measurement studies. The survey includes

Collection strategy	Questions asked	Collected answers
C_1: **All routes and subset of VPs (seven papers)**	Why did you use a subset of the VPs ?	**To speed up data processing (x2)** **For disk space and time efficiency (x1)** **I thought the rest would be similar (x1)** I did not manage to use them all (x2)
	How did you select your VPs ?	**I took them randomly (x2)** **I do not remember (x2)** It was arbitrary: my script partially failed (x1) I took geographically distant BGP collectors (x1) I did not manage to use VPs from one data provider (x1)
	Do you think more VPs would improve the quality of your results?	**Yes (x4)** Results would be similar, but it can help to find corner cases (x1) Yes, but not significantly (x1) I am not sure (x1)
	Would you have used more VPs if you could?	**Yes (x4)** **Yes, I'd love to (x1)** **Definitely (x1)** I am not sure, but I don't think so (x1)
C_2: **Limited duration of experiment (five papers)**	Was the processing time a factor that you considered when you decided on the duration of your measurement study?	**Yes (x3)**
	Do you think extending the duration of your measurement study would improve the quality of your results?	**Yes (x2)** **Yes, especially for rare events (x1)** Potentially (x1) Yes, but not significantly (x1)
	Would have extended the duration of your measurement study if you had more resources?	**Yes (x2)** Yes, but it depends on the time remaining before the deadline (x1) I think so, but also if I had more time before the deadline (x1)
All eight papers	Do you find the data from RIS and RouteViews expensive to process in terms of computational resources?	**Yes (x1)** **Yes, CPU and storage (x2)** **Yes, the storage cost and the download cost are very large (x1)** **CPU is the main issue (x1)** **RIS data takes a lot of time to download, especially when we need data for multiple days (x1)** Not the worst, but we definitely need a resourceful server if we want to catch some deadline (x1) We did that in a server so that was not a huge issue (x1) **No (x1)**
	Is there any additional challenge that you encountered when processing the BGP data from RIS and RouteViews?	**Our team used Spark clusters and Python but it was too slow (x1)** **We had to download the data from all VPs as there is no optimal solution for selecting them, the storage overhead and time overhead were extremely high (x1)** **It'll be helpful to make processing faster and less resource-consuming (x1)** Too many duplicate announcements make processing harder (x1) Variable sizes of update files exacerbate scheduling parallelization (x1) RIS took a lot longer than RouteViews (x1) We had issues when collecting updates in real-time (x1) We had to deal with bugs in BGPdump (x1) Broken data feeds and data cleanup is also an issue that we need to take care of (x1) Our study was done pre-BGPStream, which would have helped quite a bit already (x1)

Table 4: An exhaustive list of the questions asked to the participants of the survey along with their detailed answers. We color an answer in (bold) green if it (strongly) motivates the usage of a tool such as *GILL*. Blue answers are neutral, i.e., they do not motivate *GILL* but also do not disincentive it. Finally, (bold) red answers (strongly) disincentive the usage of a tool such as *GILL*.

a few questions that I will send you by email if you agree to answer them. It should take less than 5 minutes to answer it.
We might publish the results of our survey. If we do that, we will either do it in a manner that would not allow identification of your personal identity, or we will ask your permission."

17 REPRODUCIBILITY DETAILS: FINDING REDUNDANT UPDATES

We detail the algorithm used by *GILL* to find redundant BGP updates to allow reproducibility. We formalize its key functions, showcase its execution on an example, and explain how we configure its parameters. We refer the reader to §6 (component #1) for a more succinct description that focuses on the fundamental intuition and principles.

17.1 Building groups of correlated updates

Pointer: This section details *Step 1* of component #1 in §6.

Quick reminder: *GILL* builds correlation groups i.e., per-prefix sets of updates that are correlated in time. A time-correlated set of updates means that when one element of this set is observed, the other are likely to be quickly observed after. Within a correlation group, *GILL* identifies an update with its sending *VP*, *AS path*, and *community values*. Recall that all update attributes in a correlation group share the same prefix.

Example. Fig. 10 uses the same AS topology as in Fig. 5 but with four distinct events separated in time. To simplify, we only focus on prefix p1, and omit community values. This does not change how *GILL* works in practice.
Upon event #1 (time T_1): The failure on 2—4 triggers two updates, one from VP1 and one from VP2, each with an AS path that

circumvents the failure. Since these two updates for the same prefix are correlated in time, *GILL* groups their attributes into correlation group G_1.

Upon event #2 (time T_2): 2—4 is restored and VP1 (resp. VP2) receives updates that announce the primary path from 2 (resp. 6) toward p1. These two updates are different from the ones collected after the failure. Since these two updates have the same prefix and are correlated in time, *GILL* groups their attributes into correlation group G_2.

Upon event #3 (time T_3): Both 2—4 and 2—6 fail. VP1 receives an update for p1 with AS path 2—1—4. VP2 receives an update for p1 with AS path 6—3—1—4, which circumvents both failures. As these updates were not previously observed, *GILL* builds correlation group G_3 and adds their attributes.

Upon event #4 (time T_4): Both 2—4 and 2—6 are restored. VP1 (resp. VP2) receives an update that announces the primary path from 2 (resp. 6) to p1. These two updates have the same attributes as the updates collected upon event #2 and are correlated in time. *GILL* does not build a correlation group but increases by one the weight of G_2.

Settings. *GILL* has the following two parameters:

Correlation time window: Maximal time between two updates such as *GILL* considers them as correlated in time. Default is 100s to accommodate typical convergence delays [30].

Correlation groups construction time: Time during which *GILL* processes all updates for a given prefix and builds its correlation groups. The construction time must be long enough to ensure that correlation groups are representative of the actual correlation between updates received by VPs. We tested values for this parameter from one to ten days, with ten different update periods for each value. We found that after two days the ranking (in terms of weights) of correlation groups had a 94% probability of being the same as if we used another training set of the same size. This number is 95.8% when taking ten days, and 81% when taking one day. We believe that two days is a reasonable tradeoff between stability and computational expenses.

17.2 Finding redundant updates per prefix

Pointer: This section details *Step 2* of component #1 in §6.

Quick reminder. *GILL* finds redundant updates using the constructed correlation groups and an update reconstitution algorithm that relies on a new metric called the reconstitution power.

Formal definition of the reconstitution power. We denote $Corr(p, u)$ the list of correlation groups for prefix p and that includes the attributes of update u. We denote $max_{weight}(\mathcal{G}, t)$ the function that takes as input the set $\mathcal{G}$ of correlation groups, returns the update attributes included in the correlation group with the highest weight, and builds the corresponding updates by setting the timestamps to t and the prefixes to p. If multiple correlation groups have the same highest weight, $max_{weight}(\mathcal{G}, t)$ takes one of them randomly. We denote $\mathcal{U}(p, u, t)$ the set of updates reconstituted from update u with prefix p received at time t:

$$\mathcal{U}(p, u, t) = max_{weight}(Corr(p, u), t)$$

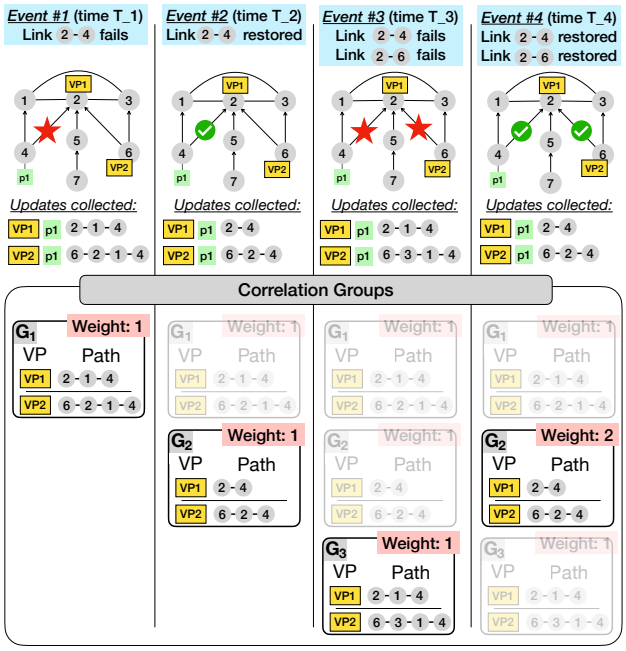


Figure 10: An example of how *GILL* builds correlation groups for prefix p1.

Consider now a set of updates β and a subset of it α, and assume that $u(t)$ is the timestamp of an update u. Consider also that two updates are identical if all their attributes are identical (*VP*, *prefix*, *AS path*, *communities*) and the difference between their two timestamps is lower than 100s. The *reconstitution power* (denoted *RP*) indicates how well we can reconstruct β from α and is defined as follows:

$$RP(\beta, \alpha) = \left| \left(\bigcup_{u \in \alpha} \mathcal{U}(p, u, u(t)) \right) \bigcap \beta \right| / |\beta|$$

Observe that $\bigcup_{u \in \alpha} \mathcal{U}(p, u, u(t))$, i.e., the set of updates reconstituted from all updates in α can include updates that are not in β. However, these incorrectly reconstituted updates (or "false positives") are ignored (operator $\bigcap$) in the *reconstitution power* definition, which only focuses on updates in β that are correctly reconstituted (the "true positive rate"). Incorrectly reconstituted updates occur when two updates u_1 and u_2 with the same *prefix*, *VP*, *AS path*, and *community values* but received at time t_1 and t_2 (with $|t_1 - t_2|$ >100s) appear correlated with distinct sets of updates, which results in u_1 and u_2 being in two different correlation groups. Thus, reconstituting updates from u_1 might result in reconstituting updates that appear with u_2 but adding to them timestamp t_1, which leads to incorrectly reconstituted updates.

Explanation of the reconstitution algorithm (with example). After building the correlation groups (§17.1), *GILL* greedily builds the set of least redundant updates α, i.e., in each iteration *GILL* adds to α the update in $\beta \setminus \alpha$ that best improves the *reconstitution power*. *GILL* adds to α either all updates received by a VP, or none because *GILL* generates filters that match on the VP and the prefix and cannot discriminate updates based on AS path or community values (see §7).

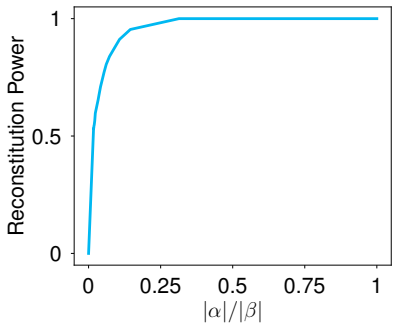


Figure 11: Reconstitution power as a function of the proportion of discarded updates ($|\alpha|/|\beta|$).

In the scenario depicted in Fig. 10, the set of initial updates β contains the eight updates induced by the four events (the i-th update is denoted U_i):

U_1: Time T_1; VP: VP1; Prefix: p1; AS path: 2 1 4
U_2: Time T_1; VP: VP2; Prefix: p1; AS path: 6 2 1 4
U_3: Time T_2; VP: VP1; Prefix: p1; AS path: 2 4
U_4: Time T_2; VP: VP2; Prefix: p1; AS path: 6 2 4
U_5: Time T_3; VP: VP1; Prefix: p1; AS path: 2 1 4
U_6: Time T_3; VP: VP2; Prefix: p1; AS path: 6 3 1 4
U_7: Time T_4; VP: VP1; Prefix: p1; AS path: 2 4
U_8: Time T_4; VP: VP2; Prefix: p1; AS path: 6 2 4

After one iteration, the reconstitution algorithm returns $\alpha = (U_2, U_4, U_6, U_8)$. The four updates all come from VP2 and enable the reconstitution of β entirely. In fact:
U_2 (in G_1) leads to the reconstitution of U_1 (also in G_1)
U_4 (in G_2) leads to the reconstitution of U_3 (also in G_2)
U_6 (in G_3) leads to the reconstitution of U_5 (also in G_3)
U_8 (in G_2) leads to the reconstitution of U_7 (also in G_2)

Observe that β cannot be entirely reconstituted if α contains the four updates collected by VP1. In fact, U_1 and U_5 have identical attribute values but appear correlated to different updates throughout time (U_1 is correlated to U_2 and U_5 is correlated to U_6). Thus, either U_2 or U_6 is not reconstituted. Besides, one update is inevitably incorrectly reconstituted. Either U_1 leads to reconstituting the following update:
Time T_1; VP: VP1; Prefix: p1; AS path: 6 3 1 4
which is not in β, or U_5 leads to reconstituting the following update (which is also not it β):
Time T_3; VP: VP1; Prefix: p1; AS path: 6 2 1 4

In practice, we observe a strong correlation in time across updates, i.e., if a set of updates appear together at time t, then the appearance of one update in this set in the future is likely to be followed by all the other updates in that set. On the RIS and RV data, we measure that among the updates that could be reconstituted but that are not in β ("negative" cases), only 4.6% are (incorrectly) reconstituted ("false positive rate").

Setting. The key parameter of *GILL* is when to stop iterating and adding new elements in α. On one hand, if $|\alpha|/|\beta|$ is close to one, the *reconstitution power* is close to one (the optimum), but at the expense of retaining many updates, leading to *GILL* building filters that retain too many updates. On the other hand, if $|\alpha|/|\beta|$ is close to zero, the *reconstitution power* is low, resulting in many nonredundant updates being discarded by the generated filters. This trade-off is visible in Fig. 11, which plots the *reconstitution factor* for different values of $|\alpha|/|\beta|$. Logically, the first updates added to α improve the *reconstitution factor* significantly. Once the *reconstitution factor* reaches 0.94, adding new updates to α has a more limited impact on the *reconstitution factor*. *GILL* thus stops iterating when the *reconstitution factor* reaches 0.94.

17.3 Finding redundant updates across prefixes

Pointer: This section details *Step 3* of component #1 in §6.

Quick reminder. BGP routes to different prefixes can be subject to similar updates. For instance, a VP likely observes the same route updates toward two prefixes announced by the same AS, which is the case for prefixes p1 and p2 in Fig. 5. Thus, the compound (for all prefixes) set of nonredundant updates returned by our algorithm (executed per-prefix) in §17.2 may include redundant updates.

Explanation of the algorithm used to find redundant updates across prefixes. *GILL* finds redundant updates across prefixes using the following algorithm: *(i)* it splits the sets of per-prefix nonredundant updates returned by our algorithm in §17.2 into distinct subsets based on the sending VP *(ii)* among all the found subsets, *GILL* identifies the ones that contain updates with identical attributes (except for the prefixes, and with a 100s slack for the timestamps) and *(iii)* for every group of identical subsets, *GILL* classifies the updates in one subset as nonredundant and the updates in the other subsets as redundant. In Fig. 5, our algorithm in §17.2 finds the same set of nonredundant updates for both p1 and p2. Thus, *GILL* classifies the set of nonredundant updates for p1 as redundant whereas the set of nonredundant updates for p2 remains classified as nonredundant.

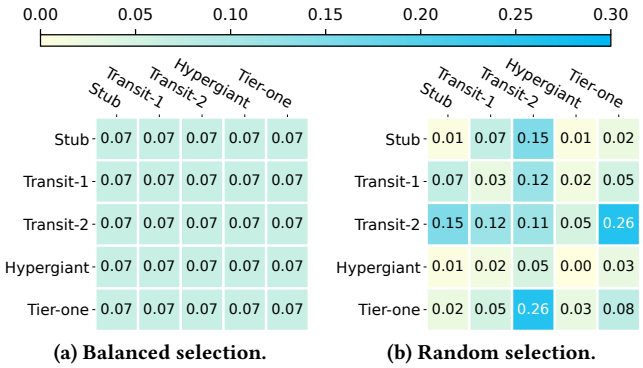
18 REPRODUCIBILITY DETAIL: SELECTING ANCHOR VPS

We explain (with formalization) the algorithm used by *GILL* to find anchor VPs and provide the methodology used to select its parameters. We refer the reader to §6 (component #2) for a more succinct description that focuses on the fundamental intuition and principles.

We consider the set of VPs V that includes all VPs from RIS and RV. We compute the RIB of VP v at time t using its last RIB dump before t and subsequent updates until t. We use this RIB to construct and maintain the directed weighted graph $G_v(t) = (N_v(t), E_v(t))$ from the AS paths of the best routes observed by v at time t, with $N_v(t)$ the set of nodes and $E_v(t) \in N_v(t) * N_v(t)$ the set of AS links. The edges are directed because two identical paths in opposite directions should not appear as redundant. Each edge in $E_v(t)$ has a weight in $\mathbb{Z}^+$ which is the number of routes in the RIB that includes this edge in their AS path.

ID	Name	# of ASes	Avg.degree	Description
1	Stub	63310	3	ASes without customer
2	Transit-1	10845	27	Transit ASes with a customer cone size lower than the average
3	Transit-2	704	267	Transit ASes ∉ Transit-1
4	HyperGiant	15	1078	Top 15 as defined in [10]
5	Tier1	19	1817	Tier1 in the CAIDA dataset [19]

Table 5: *GILL* balances selected events across AS types.



(a) Balanced selection. **(b) Random selection.**

Figure 12: *GILL* selects events using a balanced selection scheme that reduces bias. The x- and y-axes are the five categories of ASes (see Table 5).

18.1 Select BGP events to assess redundancy

Pointer: This section details *Step 1* of component #2 in §6.

Quick reminder: *GILL* uses non-global BGP events to evaluate pairwise redundancy between VPs. GILL stratifies its sample of events across space and time to avoid bias.

GILL uses local and partially visible BGP events. To assess redundancy, *GILL* focuses on BGP events that trigger topological changes: new-link events (i.e., a new link that appears in the view of at least one VP), outages (i.e., edges that disappear from the view of at least one VP), and origin changes (either legitimate or not). An event is a candidate if it has been seen by at least one VP and less than 50% of them. As mentioned in §6, *GILL* excludes global events, as it aims at finding unique pieces of data in each individual VP.

GILL avoids biases across time and location. From a candidate set of events, *GILL* builds the final set of events $\mathcal{E}$ by selecting 2250 non-overlapping events (we find that using more events does not change the performance of *GILL*), among which 750 are new-edge events, 750 are outages, and 750 are origin changes. *GILL* infers the start and end of these events by processing *all* the data that it collects using its out-of-band filtering system (described in §8).

Inspired by previous approaches to mitigate the risk of oversampling core or stub (edge) ASes [44, 57], our approach classifies ASes into five categories (Table 5) and selects an equal number of events for every pair of AS categories. The AS pair for new-link and outage events corresponds to the two ASes at both ends of the link. For origin change, it corresponds to the old and new origins. We distinguish two classes of transit providers by customer cone size (Transit-1 and -2) since they have different topological properties. If an AS belongs to more than one category, we classify it in the category with the highest ID. ASes classified in a lower row of Table 5 have a higher degree, and there are more low-degree ASes than high-degree ASes.

Fig. 12 shows the proportion of selected events for each of the 15 pairs of AS category (the matrices are symmetric) and for 2250 events selected in Sept. 2023 using two schemes: balanced and random. The random selection (Fig. 12b) selects many more events involving Transit-2 ASes (69%) than hypergiants (11%), while our balanced selection scheme mitigates biases by selecting the same number of events in every category (Fig. 12a). *GILL* selects 50 new links, outages, and origin changes in each of the 15 pairs of ASes, yielding $15 * 3 * 50 = 2250$ events ($|\mathcal{E}| = 2250$) used in next step.

18.2 Quantifying observation of VPs

Pointer: This section details *Step 2* of component #2 in §6.

Quick reminder: *GILL* evaluates how each VP experiences the selected events by computing the impact that they induce on topological features. These features embed information about *time*, *prefix*, *AS path*, and *communities*.

GILL considers the four main BGP attributes. *GILL* computes the impact of each event on the topological features [9, 22, 54] of graph $G_v(t)$ for all VPs. The combination of these topological features prevents overfitting as the graphs on which they are computed embed information about the four main BGP attributes (§2). More concretely, the graphs $G_v(t)$ embed information about (i) the *time* as the graph is built until a given time, (ii) the *AS path* as it is used to build the AS graph, (iii) the *prefixes*, used to weight every edge on the graph, and (iv) the *community values*, which strongly correlate with the AS path. We confirm this correlation by downloading the first RIBs of Sept. 2023 for all VPs and analyzing the correlation between the AS path and the set of BGP communities. We find that two identical AS paths share the exact same set of BGP communities in 93% of the cases. *GILL* thus does not embed more information about BGP communities in $G_v(t)$ because many of them encode local traffic engineering decisions [20] that could lead to overfitting. We validate this design choice in §10.

GILL uses 15 diverse topological features (Table 6). *GILL* computes topological features (extracted from literature [9, 22, 54]) that are either node-based or link-based. *GILL* computes node-based features for the two ASes involved in each event, while *GILL* computes link-based for the AS pair. *GILL* uses six node-based features that we classify into three categories. The first one quantifies how central and connected a node is; the second quantifies how connected are the neighboring nodes; and the third quantifies the topological patterns that include the node. We classify the three pair-based features into a single category that measures how close two nodes are based on their neighboring nodes. Five features rely on edge weights. We omit other topological features as they are redundant with the selected ones.

Type	Categorie	Name	Weighted	Index
Node-based	Centrality Metrics	Closeness centrality	✓	0
		Harmonic centrality	✓	1
	Neighborhood Richness	Average neighbor degree	✓	2
		Eccentricity	✓	3
	Topological Pattern	Number of Triangles	×	4
		Clustering	✓	5
Pair-based	Closeness Metrics	Jaccard	×	6
		Adamic Adar	×	7
		Preferential attachment	×	8

Table 6: Node-based and pair-based features.

GILL computes the impact of each event on the features for each VP. Consider event $e \in \mathcal{E}$ that involves two ASes e_{AS1} and e_{AS2}, starts at time e_s, and ends at time e_e. v is a VP $\in V$. Computation of the feature values depends on the feature type. We denote F_n (resp. F_p) the set of node-based (resp. pair-based) features and show how *GILL* computes the value of these two types of features for event e and VP v.

Node-based features: Consider feature $f_i \in F_n$ and $f_i(x, G_v(t))$ its value for node x on graph $G_v(t)$, with i the feature index in Table 6. *GILL* computes the following 12-dimensional feature vector.

$$\begin{aligned} T_{node\_based}(v,e) = [&f_0(e_{AS1}, G_v(e_s)) - f_0(e_{AS1}, G_v(e_e)),\\ &f_0(e_{AS2}, G_v(e_s)) - f_0(e_{AS2}, G_v(e_e)),\\ &\ldots, f_5(e_{AS1}, G_v(e_s)) - f_5(e_{AS1}, G_v(e_e)),\\ &f_5(e_{AS2}, G_v(e_s)) - f_5(e_{AS2}, G_v(e_e))] \end{aligned}$$

Pair-based features: Consider feature $f_i \in F_p$ and $f_i(x_1, x_2, G_v(t))$ its value for the node pair (x_1, x_2) on the graph $G_v(t)$, with i the feature index in Table 6. *GILL* computes the following 3-dimensional feature vector.

$$\begin{aligned} T_{pair\_based}(v,e) = [&f_6(e_{AS1}, e_{AS2}, G_v(e_s)) - f_6(e_{AS1}, e_{AS2}, G_v(e_e)),\\ &\ldots, f_8(e_{AS1}, e_{AS2}, G_v(e_s)) - f_8(e_{AS1}, e_{AS2}, G_v(e_e))] \end{aligned}$$

The final feature vector is $T(v,e)$, a 15-dimensional vector that concatenates ($\oplus$) the node- and pair-based features.

$$T(v,e) = T_{node\_based}(v,e) \oplus T_{pair\_based}(v,e)$$

18.3 Redundancy scoring

Pointer: This section details *Step 3* of component #2 in §6.

Quick reminder: *GILL* computes the pairwise redundancy scores between VPs, i.e., it computes the pairwise Euclidean distance between the feature vectors of each pair of VPs. *GILL* then computes the average pairwise Euclidean distance between each pair of VPs over all events.

Step 1: Normalize feature vectors. *GILL* normalizes the data for each event e using the feature matrix $\mathcal{M}(e)$ that includes the feature vectors for all VPs (one per row).

$$\mathcal{M}(e) = \begin{bmatrix} T(v_0, e) \\ \ldots \\ T(v_{|V|}, e) \end{bmatrix}$$

GILL normalizes (operation $\triangledown$) the matrix $\mathcal{M}(e)$ column-wise using a standard scaler that transforms every column such that its average is zero and its standard deviation is one.

Step 2: Compute Euclidean distance between VPs. *GILL* uses the normalized matrix $\triangledown(\mathcal{M}(e))$ to compute the Euclidean distance between every pair of VPs and for event e (operation $\diamond$). We denote $\triangledown(\mathcal{M}(e))_x$ the x-th row in the matrix $\triangledown(\mathcal{M}(e))$ and $\triangledown(\mathcal{M}(e))_{x,i}$ its value at index i (i.e., the i-th column). We define the Euclidean distance between the n-th VP v_n and the m-th VP v_m for event e as follows.

$$\diamond(v_n, v_m, e) = \sum_{i=0}^{15} (\triangledown(\mathcal{M}(e))_{n,i} - \triangledown(\mathcal{M}(e))_{m,i})^2$$

Step 3: Compute the average distance over all time periods. The redundancy score $\mathcal{R}(v_n, v_m)$ between two VPs v_n and v_m relates to the normalized average Euclidean distance between them over the 2250 events, computed as:

$$\mathcal{R}(v_n, v_m) = 1 - \coprod((\sum_{e \in \mathcal{E}} \diamond(v_n, v_m, e)) * \frac{1}{|\mathcal{E}|})$$

The operator $\coprod$ applies a min-max scaler so that scores are between 0 and 1, with 1 meaning the most redundant pair of VPs and 0 the least redundant pair of VPs. *GILL* thus computes and returns a redundancy score for every pair of VPs.

18.4 Generating a set of anchor VPs

Pointer: This section details *Step 4* of component #2 in §6.

Quick reminder: *GILL* selects a set of anchor VPs, considering *redundancy* and *volume*. *GILL* considers the volume of data generated by each VP as we observe that some export (sometimes significantly) more updates than others.

GILL generates the set of anchor VPs O that minimizes the proportion of redundant information collected. *GILL* initializes the set O with the most redundant VP, i.e., the one with the lowest sum of Euclidean distances to all the other VPs. This design choice allows the redundant part of the BGP data (e.g., c2p links) to be visible by the first selected VP. Thus adding VPs that have unique views is easier. At every subsequent iteration, *GILL* builds a candidate set of VPs $\mathcal{K}$ that contains the unselected VPs exhibiting the lowest maximum redundancy score. The maximum redundancy score P measures the maximum redundancy between a VP v and the set of VPs O and is defined as follows.

$$P(O, v) = \max(\mathcal{R}(v, v_i), \forall v_i \in O)$$

GILL adds to $\mathcal{K}$ the $\gamma = 10\%$ of the nonselected VPs that exhibit the lowest maximum redundancy score.

GILL then adds to set O the VP in the candidate set $\mathcal{K}$ that collects the lowest volume of data compared to the other VPs in $\mathcal{K}$. This allows *GILL* to select VPs that have a good balance between volume of collected data, and unique information added. *GILL* estimates

the volume of data collected by the VPs by counting the number of updates that they received over 365 one-hour periods, one randomly selected each day of the year to align with the yearly update rate of *GILL*'s anchor VPs (§6). The γ parameter allows tuning redundancy and volume knobs: a low γ prioritizes low redundancy while a higher γ prioritizes low resulting data volume. We found that γ = 10% performs well in practical scenarios (we tested a range from 1% to 50%). *GILL* stops adding new VPs to O when every nonselected VP has a pairwise redundancy score equal to one with at least one VP to O. With the RIS and RV VPs, we observe that the default value is 178.