

Navigating the Labyrinth: Demystifying Vulnerabilities in Routing Protocol Implementations

Yufan Zhou, Bing Shui, Hengdi Ye, Jielun Wu, and Qingkai Shi*

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

{yufanzhou, bingshui, ys.hd, jielunwu}@smail.nju.edu.cn, qingkaishi@nju.edu.cn

*Corresponding Author

Abstract—Secure networking relies on the proper implementation of routing protocols: A routing protocol implementation bug (RPIB) in a single router can rapidly cascade to other routers via routing message propagation, resulting in widespread network congestion, traffic black holes, and even significant operational and financial repercussions. Despite significant advancements, the challenges in identifying these bugs — stemming from their distributed, stateful, and timing-sensitive characteristics — allow serious vulnerabilities and attacks to persist unresolved. To systematically explore this ongoing issue, we present a comprehensive study of 1,255 real-world RPIBs from three popular open-source routing protocol implementations: FRRouting, BIRD, and Holo. From the analysis, we develop a systematic taxonomy of RPIBs, including their bug types, root causes, triggers, and security impacts, offering empirical insights to inform the future strengthening of routing infrastructures.

I. INTRODUCTION

The modern Internet is a vast, decentralized “network of networks” held together by a dense fabric of interconnected routers, which are responsible for determining the optimal path for information to travel from its source to its destination. For these routers to function as a cohesive system rather than isolated islands, they must constantly exchange information about network topology and traffic conditions. This complex coordination is governed by routing protocols, which are typically implemented as software running in routers.

Like common software, routing protocol implementations suffer from software bugs, which we refer to as routing protocol implementation bugs (RPIBs). However, unlike common software, RPIBs are quite security-sensitive: a single bug in one router can propagate through the whole network and escalate into severe damage, such as network congestion, traffic black holes, and even operational and financial repercussions. For example, in the 2020 CenturyLink outage [1], a single faulty routing update propagated across the backbone, causing a 3.5% drop in global Internet traffic. As such, RPIBs constitute a key attack vector for network-wide exploitation [2], demanding heightened attention to eliminate.

A. Motivation and Research Questions

Despite decades of effort in software development and bug detection, mature routing implementations continue to exhibit a high volume of RPIBs. For instance, FRRouting [3], an open-source routing protocol suite for Linux and Unix platforms with 1.5K forks and 4.1K stars on GitHub, consistently has >300 open issues and >300 open pull requests, with around 1/5 being RPIBs. This persistence raises an important question: why are RPIBs hard to eliminate? Addressing this requires a comprehensive analysis of existing RPIBs, which yields four research questions (RQ1-RQ4) from the perspectives of both development and quality or security assurance in routing protocol implementation.

- **RQ1 (Bug Types):** What are the predominant manifestations of RPIBs across different functional components?
- **RQ2 (Root Causes):** What are the protocol-specific, code-level root causes of these RPIBs?
- **RQ3 (Triggering Conditions):** What network environments and actions are required to trigger these RPIBs?
- **RQ4 (Security Impacts):** What security consequences could be brought by these RPIBs?

From a development perspective, developers face distinct challenges in mitigating RPIBs when implementing a routing protocol. First, RPIBs manifest heterogeneously across routing functions, from obvious crashes during packet parsing to silent security bypasses in configuration validation, so developers need a comprehensive taxonomy to recognize these diverse bug types (RQ1). Second, RPIBs have complex, protocol-specific root causes (e.g., faults in best-route calculations); fixing them requires a deep understanding of these causes (RQ2).

From the perspective of quality or security assurance, triggering and detecting RPIBs is challenging due to their highly variable triggers and elusive impacts. On the trigger side, RPIBs can be triggered by diverse protocol-specific conditions, including both proper network environments and different traffic or management actions. However, existing tools, ranging from symbolic analysis [4], [5], [6], [7], fuzzing [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], to configuration synthesis [18], [19], [20], typically focus on single, specific triggers and often neglect the effects of network environments. Thus, a comprehensive analysis of the entire trigger space is required to efficiently expose these bugs (RQ3). On the security-impact

TABLE I: Findings and Implications Drawn from the Study of Research Questions

RQs	Findings	Implications
RQ1 Bug Type	Finding 1: RPIBs can be categorized into seven functional classes. Among the seven classes, Routing, Configuration, and (De)parse bugs rank in the top three at 30.2%, 22.3%, and 17.3%, respectively, totaling about two-thirds.	It is recommended to prioritize quality assurance efforts on three critical components: route calculation, configuration handling, and packet processing.
	Finding 2: BGP and IGP implementations share similar bug types: Both the number and the distribution across all seven bug classes are largely consistent between BGP and IGP implementations.	It could be possible to develop protocol-agnostic quality-assurance techniques for both BGP and IGP, rather than spending significant effort developing them separately for each protocol.
RQ2 Root Cause	Finding 3: RPIBs' root causes can be categorized into five primary classes, each accounting for 35.2% to 9.6%, with Routing Maintenance Faults ranking first at 35.2%.	Developers must mitigate all five primary root causes during implementation, prioritizing routing maintenance, but without letting its top rank overshadow the others.
	Finding 4: Synchronization bugs frequently stem from multiple root cause classes, with no single dominant root cause, whereas other bug types are dominated by a single root cause.	The scattered root causes of Synchronization bugs make a single fix methodology impossible. Developers must adopt broad, comprehensive checking strategies, particularly for cross-protocol interactions.
RQ3 Trigger	Finding 5: Most (>98%) RPIBs can be triggered in networks with no more than three routers. Among them, networks consisting of one, two, and three routers account for 27.7%, 59.5%, and 11.0%, respectively.	While incorporating network environments into validation or verification techniques is often necessary to uncover RPIBs, a small number of routers (≤ 3) is sufficient in most cases.
	Finding 6: The two categories of actions, i.e., traffic actions and management actions, account for 39.3% and 60.7% respectively, each accounting for a substantial share.	We must consider both categories in routing validation or verification techniques, rather than predominantly focusing only on either of them.
	Finding 7: Traffic actions are predominantly driven by the injection of network traffic, which accounts for 95.8% of the observed cases. In addition, traffic flooding and interception account for only 1.9% and 2.3%, respectively.	Since injection-triggered RPIBs are essentially traffic processing flaws triggered by receiving manipulated routing data, traditional packet-oriented fuzzing techniques remain highly relevant.
	Finding 8: The four types of management actions, namely Router, Policy, Interface, and Monitor actions, occur with broadly similar frequencies, accounting for 28.0%, 27.3%, 26.9%, and 17.8%, respectively.	Bug discovery techniques involving management actions should cover all four categories of management actions rather than prioritizing a subset.
RQ4 Security Impact	Finding 9: The general security consequences of RPIBs fall into four primary dimensions: Disruption, Deception, Disclosure, and Usurpation. Overall, Disruption is the most prevalent (86.1%), while Usurpation is the least common one (1.7%).	In defending against the general security consequences caused by RPIBs, we should focus primarily on mitigating Disruption, the most prevalent consequence.
	Finding 10: All observed Usurpation cases are accompanied by other subsequent consequences, indicating that Usurpation is closely associated with broader security impacts in RPIBs.	If a security monitor detects a Usurpation event, the system should assume a total compromise is already underway. We cannot respond to the Usurpation without considering the other consequences it enables.
	Finding 11: Security consequences caused by RPIBs primarily manifest as Cut, Delay, Blackhole, and Partition in networks, accounting for 20.8%, 19.6%, 13.5%, and 13.0%, respectively, totaling about two-thirds.	When monitoring security symptoms caused by RPIBs, Cut, Delay, Blackhole, and Partition are significant indicators of compromised networks.
	Finding 12: Both the number and distribution of BGP and IGP are largely consistent across the 4 general security consequences and the 12 network-level manifestations, resulting in similar security implications.	Although BGP receives more attention in research, IGP bugs can indirectly lead to similar delivery and infrastructure damage; therefore, IGP deserves comparable security testing and research effort.

side, RPIBs often bypass crash-based detection, manifesting as silent routing-correctness violations. While methods like differential analysis [21], [22], conformance testing [23], [24], and large-language-model-based auditing [25], [26], [27], [28], [29] attempt to catch silent failures, their detection scope is restricted to a narrow range of impacts. This necessitates a systematic study of elusive impacts to help tools broadly recognize non-crashing failures (RQ4).

B. Dataset and Results

To address the aforementioned research questions, we conduct a large-scale study of 1,255 real-world bugs across three major, actively maintained open-source routing protocol suites: FRRouting [3], BIRD [30], and Holo [31]:

- FRRouting, the most popular C-based routing protocol implementation suite, is natively integrated into mainstream Linux distributions [32]. It is extensively deployed in critical backbone networks and cloud infrastructures [33] as well as industry leaders such as Nvidia [34].
- BIRD is also developed in C. It stands out as a prominent and robust routing protocol implementation suite optimized for high-performance routing environments [35].

Its deployment is ubiquitous across global Internet Exchange Points [36], earning the trust of key industry players like Cloudflare [37].

- Holo is an emerging routing protocol project designed for robust network infrastructure. Unlike the aforementioned C-based implementations, it is developed entirely in Rust, emphasizing inherent memory safety at the language level. While currently in its early stages of development, it represents a modern, memory-safe evolution in routing protocol implementations.

Our study covers five representative routing protocols across two primary categories: Interior Gateway Protocol (IGP) and Border Gateway Protocol (BGP). IGP is a class of routing protocols used to exchange routing information within a single autonomous system (AS, e.g., a corporate network). Typical IGPs include OSPF [38], IS-IS [39], RIP [40], and Babel [41]. OSPF and IS-IS employ link-state algorithms to derive shortest paths from a global view of the network, while RIP and Babel use distance-vector algorithms to determine routes based on metrics exchanged with neighboring routers.

By contrast, BGP [42], [43] is not a set of protocols but the de facto global industry standard for routing data between ASes across the Internet. For example, Fig. 1 shows a

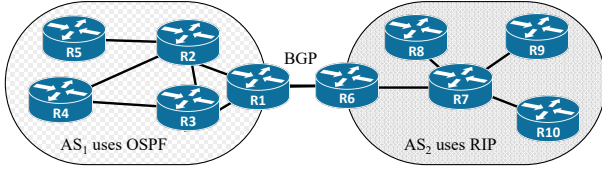


Fig. 1: IGP and BGP in Two Autonomous Systems

network with two separate ASes: one containing Routers 1–5 running OSPF, and the other containing Routers 6–10 running RIP. While these internal routers follow their respective IGP rules, the two ASes communicate via BGP to ensure global reachability.

As summarized in Table I, our comprehensive study yields 12 key findings and their implications, which are expected to offer empirical insights to inform the future strengthening of routing infrastructures. In summary, we make the following contributions in this paper:

- We build a dataset of 1,255 RPIBs across three major routing protocol implementations and five routing protocols. The dataset provides a reusable empirical foundation for routing software security, particularly including the reproducers of these RPIBs. The dataset is publicly available at <https://todiff.github.io/rpib.html>.
- We conduct a systematic empirical investigation of RPIBs across four dimensions using a routing-specific taxonomy, including bug types, root causes, triggering conditions, and security impacts.
- Through the study, we distill key findings and implications, offering actionable suggestions and directions for both developers and researchers.

II. STUDY METHODOLOGY

This section begins by discussing the assumptions (§II-A) and the threat model (§II-B), thereby establishing the scope of our study. We then explain the data collection process (§II-C) and the threats to validity (§II-D).

A. Assumptions

Behind our study, we have two widely acknowledged assumptions. First, historical bugs provide useful information about future bugs. This assumption is supported by prior studies that use historical bugs and vulnerabilities to predict new ones [44], [45], [46]. Second, following prior work [47], we assume that studying known bugs can improve software quality by reducing the number of similar bugs in the future.

B. Threat Model and Scope of Our Study

Our study focuses on bugs in protocol implementations that may lead to security consequences. First, we focus on implementation bugs; other sources of security threats, such as protocol design flaws, are out of our scope. Second, the bugs under analysis should have the potential to cause security threats. We base our threat model on the one defined in RFC

TABLE II: Manifestation of Security Consequences

Manif.	Description
Congestion	Traffic exceeds link capacity.
Blackhole	Large-volume traffic is drawn to a router that drops them.
Looping	Traffic cycles and never reaches the destination.
Partition	Routers misperceive themselves as offline.
Churn	Repeated, unnecessary routing path changes.
Instability	Routers fail to converge on consistent routing tables.
Overcontrol	Control messages (e.g., heartbeats) saturate link bandwidth.
Clog	Protocol messages exhaust device resources.
Starvation	Traffic is forwarded to a router that cannot deliver them.
Eavesdrop	Traffic is leaked to unintended observers.
Cut	Routers misperceive a reachable destination as unreachable.
Delay	Traffic is forwarded along a suboptimal path.

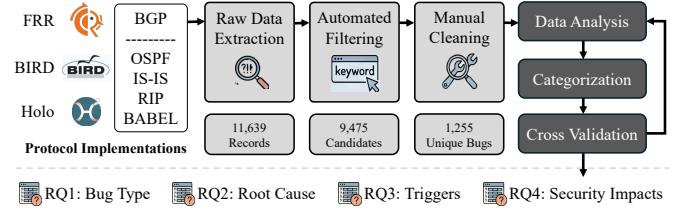


Fig. 2: Collection and Analysis Process of RPIBs

4593 [48]: “Generic Threats to Routing Protocols”. Bugs that do not cause security consequences defined in RFC 4593 are out of our scope. Specifically, section 3.1.1 of RFC 4593 defines two types of threat sources: Outsiders and Byzantine.

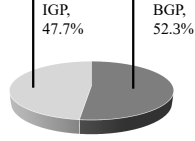
- **Outsiders** are attackers who are not legitimate participants in routing protocols. They may reside anywhere on the Internet, send IP traffic, observe the routers’ replies, and even control the path of a legitimate peer’s traffic.
- **Byzantine** are legitimate participants in the routing protocol, denoting faulty, misconfigured, or subverted routers.

The two threat sources may cause four general security consequences: Disclosure, Deception, Disruption, and Usurpation:

- **Disclosure** means routing information is exposed without authorization, either through outsiders observing insufficiently protected exchanges or Byzantine routers leaking information during legitimate protocol participation.
- **Deception** captures cases where routers accept incorrect routing information as valid, which may result from faulty or subverted Byzantine routers propagating erroneous messages, or outsiders injecting crafted messages that bypass validation.
- **Disruption** captures cases where normal routing operation is interrupted or prevented, often due to faulty or misconfigured routers entering abnormal states, or to outsiders/subverted Byzantine routers triggering crashes, session failures, floods, or invalid updates.
- **Usurpation** occurs when an attacker seizes control of a legitimate router’s services. This is executed by outsiders through the manipulation or replay of routing data, or by Byzantine routers through the injection of false information and interference with protocol exchanges.

TABLE III: Summary of the RPIB Dataset

Project	Raw	Candidates	Valid
FRRouting	7,965	7,289	830
BIRD	3,611	2,134	407
Holo	63	52	18
Total	11,639	9,475	1,255



These 4 security consequences often manifest in 12 concrete ways at the network level, including Congestion, Blackhole, Looping, Partition, Churn, Instability, Overcontrol, Clog, Starvation, Eavesdrop, Cut, and Delay. They are briefly defined in Table II. Detailed investigations and case studies are available in §VI and Appendix A.

C. Bug Collection and Analysis

As illustrated in Fig. 2, the first step is to collect bugs from three modern, actively maintained protocol implementations and five primary routing protocols, including both IGP and BGP. We then construct the RPIB dataset through a standard empirical three-stage pipeline [49], [50]: raw data extraction, automated keyword filtering, and rigorous manual verification. Ultimately, as shown in Table III, we obtain 1,255 reproducible RPIBs from an initial corpus of 11,639 records.

Data Extraction and Filtering. We collect all historical issue reports or pull requests for each routing suite from its first stable release to Jan 2026 (the time when we start our data analysis). This produces an initial corpus of 11,639 records, including 7,965, 3,611, and 63 from FRRouting, BIRD, and Holo, respectively. We then apply a keyword-based search strategy following established methodologies [51], [52], reducing the corpus to 9,475 RPIB candidates. Like prior work, the keywords include “bug”, “defect”, “error”, “fault”, “fix”, “issue”, “crash”, “security”, “correct”, and their variants. The complete keyword list is provided in Appendix B.

Manual Data Cleaning. We then manually inspect all RPIB candidates for relevance and independence. Specifically, we remove 1,586 duplicate bugs, such as patch backports and iterative fixes for the same bug, to prevent statistical bias, 3,465 documentation- or test-only bugs, 2,537 non-security bugs that do not compromise critical network security properties under our previously discussed threat model, and 632 bugs that we cannot reproduce. This yields 1,255 valid RPIBs, 52.3% from BGP implementations and 47.7% from IGP, constituting the final dataset.

Bug Analysis. Unlike traditional taxonomies that often limit bug categorization to broad “memory” versus “semantic” distinctions [53], our approach moves beyond these generalities by mapping bugs to routing-specific functional components. We specifically isolate “security-critical” RPIBs and analyze them through a multi-dimensional lens, identifying bug types, root causes, triggering conditions, and security impacts, yielding 12 actionable, domain-specific insights for practitioners. The detailed analysis is presented in RQ1 (§III), RQ2 (§IV), RQ3 (§V), and RQ4 (§VI).

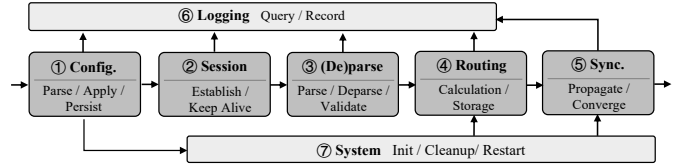


Fig. 3: Architecture of Routing Protocol Implementations

D. Threats to Validity

Like all empirical studies, our analysis may be subject to threats to validity. We outline potential threats and corresponding mitigation strategies below.

Internal validity. We ensure dataset reliability by manually verifying each of the more than 1,200 bugs. Because bug reports are written in natural language and may contain ambiguous descriptions, we use a consensus-based annotation process to reduce subjective interpretation and annotation errors. Specifically, two researchers independently code each bug, and a third arbitrator resolves any discrepancies. Before arbitration, the initial disagreement rates for RQ1–RQ4 were 12.2%, 32.0%, 26.9%, and 20.9%, respectively.

External validity. To ensure generalizability of our findings and implications, we evaluate three routing protocol implementation suites, containing both BGP and four widely deployed IGPs. We focus on open-source implementations because they provide sufficient public bug reports, whereas such data are rarely accessible for proprietary implementations. Moreover, these implementations are widely adopted by major companies, including NVIDIA and Cloudflare, showing the practical significance of our study in both the open-source community and large vendors. Since these suites are actively maintained and share fundamental design patterns with proprietary implementations, our findings are expected to generalize to modern routing environments.

Note that, as a relatively new project, Holo has far fewer bug reports than FRRouting and BIRD. We therefore do not analyze Holo separately or draw conclusions from Holo alone. All findings are based on the full set of bug reports. Also, like any repository-based empirical study, our dataset cannot capture unknown or unreported bugs. However, these limitations do not affect the conclusions drawn from the large corpus of verified bugs included in our study.

Construct validity. To accurately reflect real-world phenomena, we ground our framework in the RFC 4593 threat model. Our taxonomy captures the bug lifecycle across four dimensions: root cause, trigger, bug type, and security impact. We further mitigate misclassification risks by aligning our categories with established standards and iteratively refining the taxonomy to ensure it accurately represents the technical nuances of real-world routing vulnerabilities.

III. RQ1: BUG TYPES

Unlike traditional taxonomies that limit bug types to broad “memory” versus “semantic” distinctions, we map RPIBs to

TABLE IV: Bug Types & Distributions

Type	Sub-Type	BGP	IGP	IGP: RIP	OSPF	ISIS	BABEL
Config. 22.3%	T1 Parse Rejection	1.0%	1.3%				
	T2 Parse Corruption	1.0%	0.8%				
	T3 Parse Validation Bypass	0.3%	0.2%				
	T4 Configuration Failure	4.0%	4.0%				
	T5 Configuration Corruption	4.0%	4.1%				
	T6 Persistence Omission	0.2%	0.5%				
	T7 Persistence Malformation	0.6%	0.3%				
Session 9.6%	T8 Establishment Failure	3.0%	2.2%				
	T9 Unexpected Drop	3.1%	1.3%				
(De)Parse 17.3%	T10 Parse Rejection	0.6%	1.4%				
	T11 Parse Corruption	2.0%	1.0%				
	T12 Parse Validation Bypass	1.1%	1.9%				
	T13 Departing Failure	1.0%	1.3%				
Routing 30.2%	T14 Departing Malformation	3.9%	3.1%				
	T15 Calc. Path Omission	0.9%	2.1%				
	T16 Calc. Logic Error	3.8%	3.1%				
	T17 Storage Entry Omission	1.3%	1.0%				
Sync. 10.3%	T18 Storage Entry Corruption	9.2%	8.8%				
	T19 Propagation Failure	3.2%	3.0%				
Logging 8.9%	T20 Data Inconsistency	2.0%	2.1%				
	T21 Content Error	2.3%	1.8%				
System 1.4%	T22 Generation Failure	3.3%	1.5%				
	T23 System Management Failure	0.4%	0.6%				
	T24 System Corruption	0.1%	0.3%				

routing-specific functional components (§III-A) and analyze their distribution across different routing protocols (§III-B).

A. Taxonomy of RPIB Types

Fig. 3 shows the basic architecture of a routing protocol implementation, with seven main functional components: Configuration, Session, (De)parse, Routing, Synchronization, Logging, and System. The seven components work together to fulfill the task of a routing protocol. Our taxonomy of bug types is mapped to the seven components in the routing-specific architecture.

Walkthrough of the Architecture. In a network of interconnected routers, the workflow begins with the Configuration component, which assigns essential parameters, such as IP addresses and subnet masks, to each router interface. Once configured, routers utilize the Session component to establish stable connections with peers for the continuous exchange of routing protocol messages.

To facilitate this message exchange, the (De)parse component encodes outgoing and decodes incoming messages. These messages provide the necessary topology data for the Routing component to construct a comprehensive view of the network and build the routing table. Furthermore, the Synchronization component ensures this routing information is shared across different protocols.

Behind the scenes, two vital services support the entire lifecycle: Logging tracks component behavior, e.g., monitoring abnormal behavior (thus, we regard logging-related bugs as security-critical), while System monitors process health and intervenes to clean up or restart any failing components.

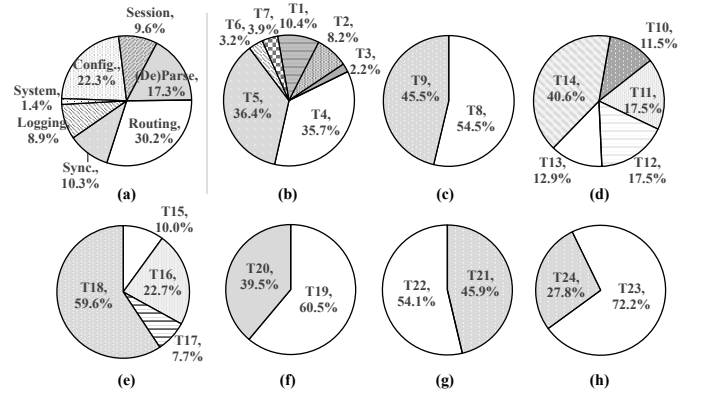


Fig. 4: RPIB (a) Type and (b-h) Sub-Type Distribution

Taxonomy and Distribution of RPIB Types. To characterize how RPIBs manifest in routing protocol implementations, we map each bug type to one of the seven functional components in Fig. 3, and then further classify a bug type into fine-grained manifestation subtypes, as listed in Table IV.

1) *RPIB Types*: As shown in the first column of Table IV and in Fig. 4(a), RPIBs are unevenly distributed across functional categories. Routing bugs are the most prevalent category, accounting for 30.2% of all bugs. Configuration and (De)parse bugs rank second and third, accounting for 22.3% and 17.3%, respectively. Synchronization bugs follow at 10.3%, while Session, Logging, and System bugs account for 9.6%, 8.9%, and 1.4%, respectively.

This distribution reflects the different roles these components play in routing protocol implementations. Routing bugs dominate because routing logic directly implements the core decision-making procedures of routing protocols, including route calculation, route selection, and route-state maintenance. The central role of these procedures makes the routing logic a natural concentration point for implementation bugs. The high proportions of Configuration and (De)parse bugs further indicate that many RPIBs arise at the boundaries between user-specified inputs and protocol messages. Configuration bugs involve the handling, validation, and persistence of user-provided parameters, while (De)parse bugs reflect the complexity of translating between protocol messages and internal data structures. Synchronization bugs rank fourth as routing implementations must correctly share routing information across different protocols. Although Session, Logging, and System bugs occur less frequently, they still affect critical supporting functions, including observability, neighbor relationship management, and process lifecycle control.

2) *RPIB Sub-Types*: Table IV further divides each RPIB type into subtypes as per how they manifest. For instance, a (De)parse bug could be a Parse Rejection bug (T10) that rejects valid protocol packets, a Parse Corruption bug (T11) that incorrectly translates packets into internal representations, and so on. Due to space constraints, we explain this in Appendix C. Overall, practitioners can refer to this distribution to benchmark the reliability of routing protocol implementations.

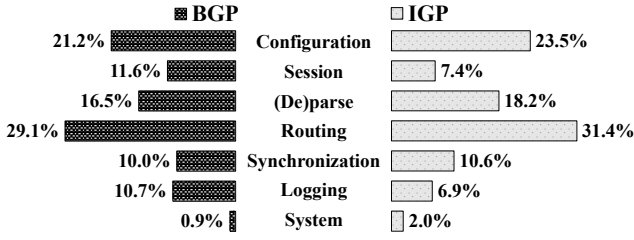


Fig. 5: Comparative Distribution of BGP and IGP

Such benchmarking allows teams to quantify how their specific implementation compares with industry-wide failure patterns, transforming raw bug counts into a comparative metric for software maturity and operational readiness.

Finding 1: RPIBs can be categorized into seven functional classes. Among the seven classes, Routing, Configuration, and (De)parse bugs rank in the top three at 30.2%, 22.3%, and 17.3%, respectively, totaling about two-thirds.

Implication: It is recommended to prioritize quality assurance efforts on three critical components: route calculation, configuration handling, and packet processing.

Since about two-thirds of RPIBs are concentrated in three functional components, development and security teams must strategically reallocate their verification resources to target these specific domains. The prominence of routing-specific bugs highlights the inherent algorithmic complexity of distributed state convergence and dynamic route calculation. This suggests a critical need for advanced testing methodologies, such as formal verification [54] or specialized state-space exploration [55], which go beyond traditional unit testing to proactively identify edge cases in networks.

Meanwhile, the high ratio of (de)parse and configuration bugs exposes severe weaknesses in how protocol implementations process external, untrusted inputs. (De)parse components act as the critical boundary between the network and the application logic; their high error rate strongly underscores the necessity for rigorous, protocol-aware fuzzing to guarantee memory safety and strict RFC compliance against malformed packets [25], [56]. Similarly, the prevalence of configuration-related faults points to the underlying fragility of managing vast, interconnected router states. To meaningfully reduce these bugs, future efforts may prioritize developing robust, automated configuration validation frameworks, e.g., ToD-IFF [22], that can safely isolate and verify configurations before applying them to the live routing environment.

B. Bug Distribution across Routing Protocols

Distribution across IGP. The last column of Table IV illustrates the relative proportions of four mainstream IGPs (i.e., RIP, OSPF, IS-IS, and BABEL) across seven bug types and their corresponding sub-types. From both aggregate and per-type perspectives, OSPF contributes the highest number of bugs, followed by IS-IS, while BABEL and RIP account for the smallest shares. While OSPF accounts for more than half of the bugs in most subtypes, several anomalies exist: for

instance, IS-IS is responsible for half of T7 bugs, and BABEL accounts for nearly 40% of T12 bugs.

These outliers reflect specific implementation maturity levels of each IGP’s components. For example, the prevalence of T12 (Parse Validation Bypass) in BABEL suggests that its input validation logic for certain message types is less rigorous than that of more legacy, battle-tested protocols such as OSPF. In practice, when developers or network administrators adopt a specific IGP, the distribution in Table IV could guide us to pay more attention to a fragile component (e.g., T12) within that IGP (e.g., BABEL).

BGP vs. IGP. By aggregating the sub-type distributions of BGP/IGP in Table IV, Fig. 5 presents the bug-type distribution for BGP and IGP implementations, respectively. Despite significant design differences between the two protocol families, they exhibit a very similar distribution of bugs across all seven bug classes. Routing bugs and Configuration bugs are consistently the most common categories in both protocols (Routing: BGP 29.1%, IGP 31.4%; Configuration: BGP 21.2%, IGP 23.5%), indicating that managing routing state and handling configuration complexity are shared sources of bugs across protocols. On the other end, System bugs remain the least frequent in both cases (System: BGP 0.9%, IGP 2.0%), suggesting that this aspect is relatively well-handled in practice.

Recall that BGP and IGP have a similar number of bugs in the dataset (see Table III). Thus, the distribution comparisons below also reflect the absolute bug counts.

Finding 2: BGP and IGP implementations share similar bug types: Both the number and the distribution across all seven bug classes are largely consistent between BGP and IGP implementations.

Implication: It could be possible to develop protocol-agnostic quality-assurance techniques for both BGP and IGP, rather than spending significant effort developing them separately for each protocol.

Despite operating in vastly different scopes and environments, both BGP and IGP suffer from similar classes of RPIBs. For instance, Configuration, (De)parse, and Routing are the most vulnerable components regardless of whether the protocol is IGP or BGP. Thus, engineers and researchers may abstract the underlying routing logic to build unified, protocol-agnostic quality-assurance frameworks for both IGP and BGP. That is, instead of dedicating resources to develop specialized static analyzers, fuzzers, or model checkers for BGP (e.g., [57], [58]) and IGP variants (e.g., [22], [59]) independently, we may design tools around a generalized routing protocol model. This cross-pollination could significantly reduce engineering overhead and allow the community to develop robust, overarching quality assurance methodologies that simultaneously fortify the broader routing ecosystem.

IV. RQ2: ROOT CAUSES

The preceding taxonomy characterizes where a bug surfaces in the routing workflow. This section introduces an orthogonal

TABLE V: Taxonomy of RPIB Root Causes

ID	Root Cause Description	Distribution
C1	I/O Faults	 28.6%
C2	Routing State Machine Faults	 9.6%
C3	Routing Calculation Faults	 10.4%
C4	Routing Maintenance Faults	 35.2%
C5	Resource Management Faults	 16.2%

causation axis capturing which correctness contract is violated at the code level. These dimensions are conceptually distinct: a root cause can manifest in functionally disparate components, and a functional component can be corrupted by multiple root causes. To this end, we first establish a taxonomy of root causes (§IV-A) and then analyze the relationship between RPIB types and their underlying root causes (§IV-B).

A. Taxonomy of RPIB Root Causes

Following Avizienis et al. [60], we define a root cause as a dormant fault, i.e., a code-level correctness contract violation. Routing protocol bugs arise from two scenarios: violations of protocol specifications or improper use of protocol-independent system resources (e.g., memory, network sockets). As shown in Table V, we categorize root causes into five types: C1–C4 represent specification violations, while C5 covers resource management.

C1: I/O Faults. Routing protocols process external data (e.g., raw protocol packets, configuration commands) via encoding and decoding. Failures to correctly process these data per the specification are categorized as C1. For instance, Fig. 6(a) shows an input validation bug when parsing a special OSPF packet, denoted by the variable `lsa` at line 6. The packet includes an Advertiser-ID field (`lsa->adv_id`). This field is invalid when set to `0.0.0.0`. In such cases, the `get_edge` function at line 9 returns a null pointer, leading to a null pointer dereference at line 12. The patch addresses this by adding a null pointer check at line 10. Fig. 6(b) illustrates a configuration file parsing bug within the RIP protocol implementation. The original implementation attempts to read a “Path” entry, i.e., denoted by `paths` at line 7, from the argument `argv[1]` due to the default value of `idx_number` is 1 at line 3. However, this is not necessarily the case. The patch fixes this by adding an `argv_find` operation to obtain the correct `idx_number` at line 6.

C2: Routing State Machine Faults. Routing protocols are often stateful, controlled by state machines to fulfill tasks such as link initialization, message exchange, etc. Faults that violate state machine rules specified in protocol specifications are categorized as root cause C2. For example, Fig. 7 illustrates an RPIB rooted in a C2 fault. Specifically, when verifying link activity between two routers, the original implementation returns false whenever the session state is down (line 4 through `bfd_session_is_down()`), thereby determining the link as inactive and terminating it. While this seems reasonable, it fails to account for links during the initialization phase, when the session state is naturally down, and the implementation

```

01. void ospf_te_parse(ospf_packet *pkt, ospf *ospf) {
02.     // Parse packet only when TE is enabled.
03.     if (!CHECK_FLAG(ospf->flags, TE_ENABLE)) { return; }
04.     // ...
05.     // Malformed LSA with invalid advertiser_id 0.0.0.0
06.     ospf_lsa *lsa = extract_lsa(pkt);
07.     // ...
08.     // get_edge return NULL due to invalid advertiser_id
09.     ls_edge *edge = get_edge(lsa->adv_id, ...);
10. + if (!edge) { return; }
11.     // NULL pointer dereference
12.     ls_attributes *attr = edge->attributes;
13.     // ...
14. }
(a)

01. int rip_allow_ecmp(int argc, cmd_token *argv[]) {
02.     // ...
03.     int idx_number = 1;
04.     uint32_t paths = 0;
05. - if (argv[idx_number])
06. + if (argv_find(argv, argc, "Path", &idx_number))
07.     paths = strtoul(argv[idx_number]->arg, NULL, 10);
08.     // ...
09. }
(b)

```

Fig. 6: Code Examples for C1 (PR#15674, PR#13578)

```

01. bool peer_active(peer *peer) {
02.     // ...
03.     if (peer->bfd_config) {
04. - if (bfd_session_is_down(peer->bfd_config->session))
05. + if (peer_established(peer->connection) &&
06. +     bfd_session_is_down(peer->bfd_config->session))
07.         return false;
08.     }
09.     // ...
10. }

```

Fig. 7: Code Example for C2 (PR#18682)

terminates links that have not yet been established. The patch fixes this by adding a condition at line 5 that checks whether the link is in the initialization phase.

C3: Routing Calculation Faults. Routing protocols use algorithms, such as Dijkstra and Bellman-Ford algorithms, to calculate/select the shortest/best routes in a network. Bugs resulting from incorrect calculation results are categorized as C3. For example, Fig. 8 illustrates a bug in FRRouting’s OSPF implementation rooted in the erroneous calculation of the routing cost. RFC 5340 [61] explicitly mandates a cost of zero for loopback interfaces. The original implementation does not include a special check of loopback interfaces at line 5 and, thus, returns a non-zero cost for a loopback interface. As such, all subsequent shortest-route calculations use this incorrect cost, ultimately resulting in an incorrect routing table. The patch addresses this by initializing the cost to zero (line 3) and introducing a fast-path check (lines 5–6).

C4: Routing Maintenance Faults. Unlike C3, which focuses on the calculation process, C4 considers faults in internal structures (e.g., link-state databases, neighbor structures) maintained to support route calculation in a dynamic network environment. For example, given that Link State Advertisements (LSAs) are specialized packets stored internally to maintain routing information, Fig. 9 illustrates a bug where a stale LSA packet is not purged from the link-state database within an OSPF protocol implementation. When removing external

```

01. uint32_t ospf_interface_get_cost(ospf_interface *oi) {
02. - uint32_t cost;
03. + uint32_t cost = 0;
04. // ...
05. + if (if_is_loopback(oi->interface))
06. + return cost;
07. // Calculate default interface cost (non-zero).
08. //...
09. }

```

Fig. 8: Code Example for C3 (PR#14635)

```

01. void ospf6_external_lsa_purge(ospf6 *ospf6,
02.                               ospf6_lsa *lsa) {
03. //...
04. uint32_t lsa_id = lsa->header->id;
05. uint32_t ospf_id = ospf6->router_id;
06. // Delete TYPE-5 LSA
07. ospf6_lsa_purge(lsa);
08. + // Delete TYPE-7 LSA
09. + for (ALL_LIST_ELEMENTS(ospf6->area_list)) {
10. + lsa = ospf6_lsdb_lookup(TYPE_7, lsa_id, ospf_id);
11. + if (lsa)
12. + ospf6_lsa_purge(lsa);
13. + }
14. }

```

Fig. 9: Code Examples for C4 (PR#9342)

LSAs, the original implementation removes only the Type-5 LSA at line 7, omitting Type 7. The patch addresses this by explicitly deleting the Type-7 LSA (lines 8–13).

C5: Resource Management Faults. These faults are independent of protocol semantics and stem from the improper management of low-level system resources. For example, Fig. 10 illustrates an RPIB in an IS-IS protocol implementation caused by incorrect memory management. During routing calculations, a dynamically allocated topology vertex (line 3, representing a router) is concurrently referenced by a processing queue and a lookup hash table. When the original implementation discards an obsolete vertex, it correctly removes the vertex from the queue at line 5 and deallocates memory at line 7, but fails to purge the vertex from the hash table. This results in a dangling pointer, triggering a use-after-free vulnerability during subsequent lookups. The patch addresses this by explicitly removing the vertex from the hash table at line 6.

About one-third of the bugs with root cause C5 (accounting for 5.3% of all bugs) are memory-safety issues, such as use-after-free and double-free bugs. These issues can be prevented at compile time with Rust, as in Holo.

Distribution of Root Causes. As shown in Table V, we employ this taxonomy to analyze the distribution of root causes. Routing Maintenance Faults (C4) are the most prevalent, accounting for 35.2% of the dataset, followed by I/O Faults (C1) at 28.6%. In contrast, Routing State Machine Faults (C2) are the least frequent, representing 9.6%. While every category represents a non-negligible share (near or above 10%), C1 and C4 clearly dominate, together accounting for 63.8% of all faults. This predominance stems from the inherent operational mechanics of routing protocols. At their core, routing daemons act as complex, continuous data-processing pipelines: they

```

01. void processN(isis_spftree *spf, uint32_t id) {
02. isis_spftree *vertex;
03. vertex = isis_find_vertex(&spf->tents, id);
04. // ...
05. isis_vertex_queue_delete(&spf->tents, vertex);
06. + hash_release(spf->prefix_sids, vertex);
07. isis_vertex_del(vertex);
08. // ...
09. }

```

Fig. 10: Code Example for C5 (PR#16021)

must constantly parse loosely formatted external packets from untrusted sources, while simultaneously maintaining massive, highly dynamic internal data structures, such as routing tables and neighbor topologies.

Because no single category accounts for a vast majority of the bugs, developers cannot rely on a one-dimensional testing strategy. Instead, this distribution implies a critical need for a multi-faceted, defense-in-depth approach to quality assurance. To secure the external attack surface against C1 faults, developers should aggressively integrate grammar-aware fuzzing [12], [25] to validate boundary conditions and input sanitization. For internal semantic correctness, such as mitigating C2, C3, and C4 faults, developers are encouraged to adopt stateful analysis to strictly enforce state machine invariants [10], [56]. Finally, resource management faults (C5) should be systematically prevented by incorporating standard static analyses, fuzzers, and memory sanitizers (e.g., ASan [62]) directly into the continuous integration pipeline.

B. RPIB Types vs. Root Causes

RPIB types discussed in §III-A and code-level root causes are orthogonal by design. To explore their correlations, we cross-map all seven bug types against the five root causes.

Causal Locality. As shown in Fig. 12, six out of the seven RPIB types are dominated by a single fault category: C1 causes 80.7% of (*De*)parse bugs and 57.7% of *Logging* bugs; C2 drives 52.1% of *Session* bugs; C4 accounts for 58.2% of *Configuration* bugs and 41.4% of *Routing* bugs; and C5 triggers 83.3% of *System* bugs. This concentration indicates strong causal locality: bugs within a specific component typically originate from root causes closely associated with that component’s core functionality. For example, since (*De*)parse components are primarily composed of encoding and decoding logic, they are inherently tied to I/O Faults (C1); consequently, the root cause of (*De*)parse bugs is most frequently C1.

Nonetheless, such causal locality is not absolute. Root causes that appear logically distant from a specific bug type can still be the underlying source. For example, Fig. 11 shows a *Deparsing* bug in a BGP implementation rooted in Routing Maintenance Faults (C4) instead of I/O Faults (C1). Specifically, the buggy code incorrectly maintains an internal data structure, namely `peer->flags`: it omits the flag-setting at line 4. As such, the deparsing procedure (lines 8-15) generates a malformed packet because it skips the branch at line 11 and fails to append a flag to the outgoing packet at line 12.

```

01. int set_peer_interface(peer *peer) {
02.     // set peer flags
03.     SET_FLAG(peer->flags, CAPABILITY_ENHE);
04. + SET_FLAG(peer->flags, CAPABILITY_LINK_LOCAL);
05.     // set other flags ...
06. }
07.
08. uint16_t bgp_gen_peer_packet(peer *peer) {
09.     uint8_t *packet;
10.     // ...
11.     if (CHECK_FLAG(peer->flags, CAPABILITY_LINK_LOCAL)){
12.         append_data(packet, CAPABILITY_CODE_LINK_LOCAL);
13.     }
14.     // ...
15. }

```

Fig. 11: (De)parse Bugs Rooted in C4 (PR#18080)

Finding 3: RPIBs’ root causes can be categorized into five primary classes, each accounting for 35.2% to 9.6%, with Routing Maintenance Faults ranking first at 35.2%.

Implication: Developers must mitigate all five primary root causes during implementation, prioritizing routing maintenance, but without letting its top rank overshadow the others.

An Exception. As an exception, Synchronization bugs, which concern cross-protocol interactions, are not dominated by a single root cause. Instead, they are affected by all five root causes to a roughly comparable extent: C1 (26.4%), C2 (16.3%), C3 (10.8%), C4 (33.3%), and C5 (13.2%). This is because the synchronization component in a routing protocol implementation serves as a bridge across multiple protocols, which requires the correct functioning of multiple domains, including packet processing (closely associated with C1), state management (C2), path computation (C3), internal data structure maintenance (C4), and resource management (C5). Since a violation in any of these areas can manifest as a synchronization bug, these bugs are inherently rooted in a wide spectrum of root causes.

Finding 4: Synchronization bugs frequently stem from multiple root cause classes, with no single dominant root cause, whereas other bug types are dominated by a single root cause.

Implication: The scattered root causes of Synchronization bugs make a single fix methodology impossible. Developers must adopt broad, comprehensive checking strategies, particularly for cross-protocol interactions.

To secure the synchronization component, developers cannot rely on localized, single-component testing methodologies or targeted fixes. Instead, they must adopt holistic, cross-layer verification or validation strategies that evaluate the synchronization component as an integrated system. In particular, we shall design and develop quality-assurance techniques, e.g., [63], to address cross-protocol interactions, which frequently result in critical, hidden consequences [64], [65]. While research in this area is growing, it still lags behind the more common focus on single-protocol analysis.

V. RQ3: TRIGGERING CONDITIONS

Triggering a bug is a fundamental prerequisite for its confirmation, remediation, and exploitation. Different from conventional software, triggering an RPIB necessitates a complex

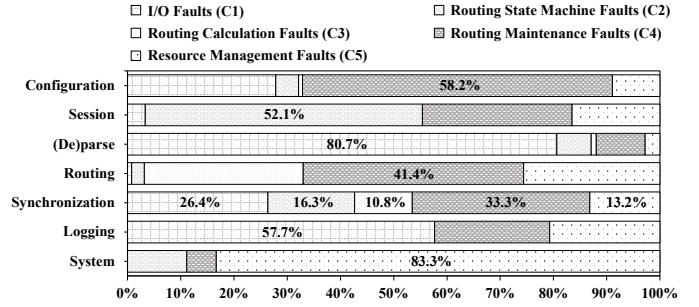


Fig. 12: Root Causes vs. RPIB Types

interplay of environmental factors due to the decentralized nature of routing protocols. We define an RPIB trigger as a tuple $(Env, Action)$, where Env represents a network environment, containing a set of interconnected routers (§V-A), while $Action$ denotes the inputs (§V-B), including traffic actions (§V-C) and management actions (§V-D). This framework acknowledges that many RPIBs are triggered only when an appropriate action is performed within a specific network.

A. Network Environment

Establishing a network environment (i.e., a network topology with interconnected routers) is often a prerequisite for triggering an RPIB. Although the complexity of the network environment can be characterized by various metrics, we use “the minimal number of routers required to trigger an RPIB” as our classification metric because it is simple, clear, and directly reflects the network scale. Specifically, we first construct a “large” topology that triggers the RPIB and then iteratively remove routers until any further removal prevents its reproduction, yielding the minimal number of routers.

Surprisingly, despite the decentralized and distributed characteristics of routing protocols, as shown in Fig. 13(a), almost all RPIBs (>98%) can be triggered in networks consisting of no more than three routers, with only the remaining 1.8% requiring over three routers. Two-router networks account for the largest proportion (59.5%), followed by single-router networks (27.7%). Combined, these two categories constitute more than four-fifths of all cases, suggesting that most RPIBs can be triggered either locally on a single router or through interactions between neighboring routers. Three-router networks account for a further 11.0% of cases, mainly because routing-path bugs often arise only when the buggy router must choose among routes learned from at least two competing next-hop routers, and three routers are often sufficient to create such route-selection scenarios.

Finding 5: Most (>98%) RPIBs can be triggered in networks with no more than three routers. Among them, networks consisting of one, two, and three routers account for 27.7%, 59.5%, and 11.0%, respectively.

Implication: While incorporating network environments into validation or verification techniques is often necessary to uncover RPIBs, a small number of routers (≤ 3) is sufficient in most cases.

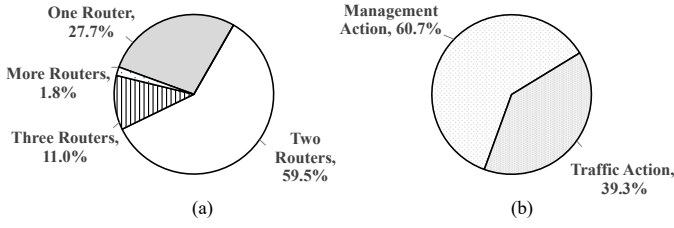


Fig. 13: (a) Distribution of #Routers; (b) Management Actions vs. Traffic Actions

While creating large-scale topologies containing dozens of routers could make it easy to find bugs, such as [66], [67], our analysis above shows that most RPIBs can be triggered in a network with no more than three routers. This result implies that large-scale environments may not always be necessary for effective RPIB discovery. Thus, existing techniques could be improved in small-scale network environments (that are simpler and less costly) while remaining highly effective.

B. Traffic Actions vs. Management Actions

Beyond the network environment, triggering an RPIB requires specific traffic and management actions. While a sequence of actions may be involved to trigger a bug, we focus exclusively on the final action that directly executes the buggy code. Preceding actions merely establish prerequisites; therefore, our analysis centers on the terminal action that holds a direct causal link to the RPIB.

Traffic Actions. RPIBs may be triggered by traffic actions, such as sending a malformed packet to a router or overloading a router with excessive traffic, among others. Fig. 6(a) shows a typical example in which, upon receiving a special packet, the router executes the buggy function `ospf_te_parse`, thereby triggering the RPIB. Additionally, the bugs in Fig. 9 and Fig. 10 can both be triggered by sending routing update packets; specifically, the former is triggered by an update packet that forces the removal of an external router’s status, whereas the latter is triggered by a packet that drops a neighboring router from a routing table.

Management Actions. RPIBs may be triggered by management actions, such as configuring routing policies, updating link costs, or adjusting process controls, which are often performed via command-line interfaces by privileged users. Fig. 6(b) shows a typical example: when an administrator configures the number of optimal paths for the RIP protocol (e.g., via the command `router rip; allow-ecmp [num]`), it runs the buggy function and triggers the bug at line 5. Furthermore, the bugs in Fig. 7, Fig. 8, and Fig. 11 are triggered by management commands that establish a new link, retrieve interface costs, and configure interface attributes, respectively.

Distribution. Fig. 13(b) shows that both action categories account for a substantial share of RPIB triggers: management actions trigger 60.7% of RPIBs, and traffic actions trigger the remaining 39.3%. Although management actions are more common, neither category is negligible. This is expected, since

routing implementations maintain correct routing state through both dynamic network interactions (e.g., processing protocol packets) and administrative interventions (e.g., applying configurations and policies). Both inputs are therefore significant attack surfaces, and testing that focuses on only one of them would miss a large fraction of RPIBs.

Finding 6: The two categories of actions, i.e., traffic actions and management actions, account for 39.3% and 60.7% respectively, each accounting for a substantial share.
Implication: We must consider both categories in routing validation or verification techniques, rather than predominantly focusing only on either of them.

While both traffic and management actions are critical in triggering RPIBs, current mainstream research predominantly focuses on leveraging traffic actions for routing testing (e.g., via traffic flooding [68], [69], packet-oriented fuzzing [25], [56], and dynamic packet interception [70], [71]), with significantly less attention directed at management actions. This finding suggests that future routing testing techniques should seamlessly integrate management actions and systematically explore the vast and intricate state space introduced by administrative configurations to uncover more RPIBs.

C. Breakdown of Traffic Actions

We classify traffic actions into three categories based on how network traffic can be manipulated to trigger RPIBs: Injection, Flooding, and Interception. The former two denote actions in which attackers or peers send packets to a target router, whereas the latter denotes cases in which attackers intercept packets sent from a target router.

Traffic Injection. Traffic Injection means an RPIB is triggered when an attacker or a peer sends the target router one or a small number of routing packets. The packet may be well-formed, malformed, forged, content-modified, or part of a small-scale replay attack. This is the most common case, accounting for 95.8% of the observed RPIB triggers. The bugs discussed before, i.e., Fig. 6(a), Fig. 9 and Fig. 10, belong to this case.

Traffic Flooding. Unlike traffic injection, which triggers bugs by sending a small number of routing packets, Flooding occurs when a router’s flawed logic fails under high traffic stress. This is triggered when an attacker or peer sends routing packets at a massive scale, overwhelming the device. For example, an OSPFv3 bug in FRRouting [PR#9019](#) caused unnecessary scans of unrelated routes when processing many thousands of routes. The resulting prolonged SPF processing could disrupt adjacencies configured with short dead timers.

Traffic Interception. Unlike Injection and Flooding, which rely on sending malicious or excessive traffic to a target router, Interception intercepts, drops, blocks, or filters packets sent out from the target router. Such interception may trigger fault-tolerant mechanisms, such as resending a dropped packet, thereby leading to certain RPIBs. For example, in FRRouting [PR#15026](#), an OSPF bug is triggered when OSPF Hello

packets are blocked. This interception causes the connection between routers to time out and disconnect. Once the router detects the disconnection, it activates its fault-tolerant mechanism, which fails to handle the sudden disconnection and retains outdated routing data in its database, leading to inconsistent network information.

Distribution. Fig. 14(a) shows that traffic actions are dominated by injection, accounting for 95.8% of the observed RPIB triggers in this category, while the others, such as flooding and interception, only account for 1.9% and 2.3%, respectively. The rarity of other actions is logical: routing protocols naturally tolerate dropped packets and missing acknowledgments, mitigating the impact of interception. Besides, flooding targets implementation flaws under high stress, which are less common as they are robustness issues rather than logic bugs.

Finding 7: Traffic actions are predominantly driven by the injection of network traffic, which accounts for 95.8% of the observed cases. In addition, traffic flooding and interception account for only 1.9% and 2.3%, respectively.

Implication: Since injection-triggered RPIBs are essentially traffic processing flaws triggered by receiving manipulated routing data, traditional packet-oriented fuzzing techniques remain highly relevant.

This finding highlights an alignment between the nature of RPIBs and the capabilities of standard security testing tools. Injection dictates that an RPIB is triggered when a router receives and processes packets from an external source. Since conventional packet-oriented fuzzers, e.g., [8], [9], [10], natively simulate this exact interaction by establishing peer connections and injecting mutated data, their testing methodology mirrors the dominant RPIB trigger.

D. Breakdown of Management Actions

Following the documentation of the investigated routing protocol implementations [72], [73], we divide management actions into four categories: Router, Interface, Policy, Monitor.

Router Actions. Router actions bring routers up or down and define global routing attributes, such as the unique router identity for establishing connections, and the hello/dead timers for keeping connections alive, among many others. For example, `router ospf; ospf router-id 1.1.1.1` is a router action that sets the identity of a router to 1.1.1.1.

Interface Actions. Each router may contain multiple interfaces connecting to other routers. Interface actions define the attributes of interfaces and their associated links (e.g., interface IP addresses and link costs). For instance, `interface eth0; ip ospf cost 100` is an interface action that sets the link cost on interface eth₀ to 100, which is used to compute routing paths.

Policy Actions. Policy actions configure routing strategies to determine feasible routing paths, e.g., by excluding specific paths from selection. For example, `router rip; allow-ecmp 2` is a policy action for RIP that enables a router to forward packets over up to two equal-cost paths at the same time.

Monitor Actions. Monitor actions are used to inspect the runtime status for maintenance and debugging purposes. For

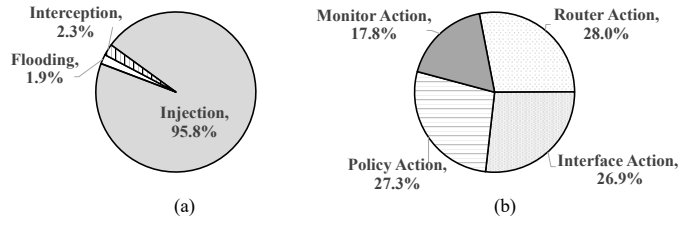


Fig. 14: (a) Breakdown of Traffic Actions; (b) Breakdown of Management Actions

instance, `show bgp neighbor` is a BGP monitor command that allows the administrator to view all neighbors connected to a router.

Distribution. The distribution of the four kinds of management actions is shown in Fig. 14(b). Router actions rank first, accounting for 28.0%, followed closely by Policy actions at 27.3% and Interface actions at 26.9%. Monitor actions account for the smallest share, at 17.8%. Overall, no single type of management action is dominant.

Finding 8: The four types of management actions, namely Router, Policy, Interface, and Monitor actions, occur with broadly similar frequencies, accounting for 28.0%, 27.3%, 26.9%, and 17.8%, respectively.

Implication: Bug discovery techniques involving management actions should cover all four categories of management actions rather than prioritizing a subset.

Leveraging the management actions for testing (or triggering RPIBs) is challenging because different action types require distinct testing strategies. Specifically, Monitor actions can reveal routers' internal states and thus align well with black-box testing. In contrast, Router and Interface actions can alter connectivity between neighbors, requiring session-level testing oracles, whereas Policy actions can affect route selection and propagation, requiring routing-result oracles. As a result, comprehensive testing must integrate multiple techniques and specialized oracles to test all action types.

VI. RQ4: SECURITY IMPACTS

Returning to the threat model this work applies, this section examines the security impacts that the RPIBs may produce. Particularly, as discussed in §II-B, each security impact can be viewed from two complementary angles: the general security consequences and their manifestation in networks. We first analyze these two dimensions in §VI-A and §VI-B, and then compare IGP and BGP in §VI-C.

A. General Security Consequences.

As detailed in §II-B and RFC 4593 [48], RPIBs can cause four general security consequences: Disclosure, Deception, Disruption, and Usurpation. Fig. 15 shows their distribution from different angles.

Overall Distribution. Fig. 15(a) reports the distribution over all observed consequences: an RPIB that produces multiple consequences contributes once to each of them, so the

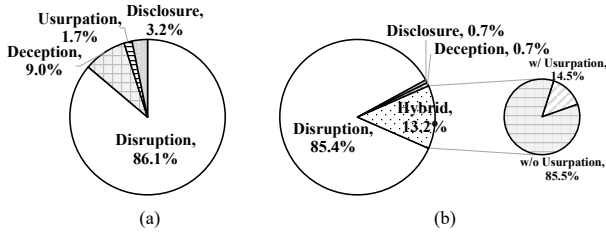


Fig. 15: Distribution of Security Consequences

percentages are shares of consequence occurrences rather than of bugs. As shown in (a), Disruption (i.e., the interruption or degradation of routing services) dominates at 86.1%, largely because routing implementations operate as a sequential pipeline as illustrated in Fig. 3. Therefore, an error in any intermediate stage, such as Configuration, (De)parse, or Routing, can potentially cause incorrect routing calculations or even complete protocol crashes, directly leading to Disruption.

Conversely, Usurpation (i.e., seizing unauthorized control of a legitimate router’s services) accounts for the smallest share (1.7%). While implementation errors frequently disrupt packet forwarding or protocol sessions, successfully subverting protocol control or policy execution to gain unauthorized service control requires highly specific logic flaws, making Usurpation much rarer.

Finding 9: The general security consequences of RPIBs fall into four primary dimensions: Disruption, Deception, Disclosure, and Usurpation. Overall, Disruption is the most prevalent (86.1%), while Usurpation is the least common one (1.7%).

Implication: In defending against the general security consequences caused by RPIBs, we should focus primarily on mitigating Disruption, the most prevalent consequence.

Given that Disruption dominates the general security consequences, accounting for over four-fifths of the observed impacts, defense strategies must urgently prioritize maintaining routing service availability. This strongly suggests a critical need to integrate established routing disruption detection techniques, e.g., [74], [75], into the network monitoring ecosystem. By leveraging these techniques, we can continuously detect the early symptoms of Disruption caused by RPIBs, thereby enabling timely repair and strengthening the resilience of the overall routing infrastructure.

Hybrid Consequences. Fig. 15(b) captures the fact that an RPIB may produce multiple security consequences. It reports the distribution over bugs: each RPIB is counted exactly once, either under the single consequence it produces or under *Hybrid* if it produces multiple (hybrid) consequences. We have three observations from Fig. 15(b). First, 85.4% of RPIBs produce Disruption in isolation, which lets Disruption still rank first even from a different angle. Second, 13.2% of RPIBs cause hybrid consequences, indicating that RPIBs can often trigger a “compound threat” in which a single implementation flaw simultaneously compromises multiple security pillars. Third, Usurpation does not occur in isolation, appearing in 14.5% of hybrid cases. This is because once an

attacker controls a router’s services, it can readily misdirect (Deception), expose (Disclosure), or interrupt (Disruption) the traffic those services handle.

Finding 10: All observed Usurpation cases are accompanied by other subsequent consequences, indicating that Usurpation is closely associated with broader security impacts in RPIBs.

Implication: If a security monitor detects a Usurpation event, the system should assume a total compromise is already underway. We cannot respond to the Usurpation without considering the other consequences it enables.

Since Usurpation inherently triggers cascading malicious impacts, security monitors must treat it as an indicator of a broader, multi-stage attack rather than an isolated anomaly. Mitigating this threat within decentralized routing architectures requires a comprehensive approach across detection, response, and containment. To limit the scope of consequences, frameworks like RPKI [76], [77] can reject advertisements with invalid origins, limiting the extent to which a compromised node can propagate deceptive routes.

B. Manifestation in Networks

We further study how the security consequences caused by RPIBs manifest at the network level. As listed in Table II and detailed in Appendix A, RFC 4593 [48] categorizes these manifestations into 12 ways: Congestion, Blackhole, Looping, Partition, Churn, Instability, Overcontrol, Clog, Starvation, Eavesdrop, Cut, and Delay.

As shown in Fig. 16, Cut (i.e., misperceiving a reachable destination as unreachable), Delay (i.e., forcing traffic onto suboptimal paths), Blackhole (i.e., attracting traffic to a router that drops it), and Partition (i.e., misperceiving themselves as offline) are the four most frequent network-level damages, accounting for 20.8%, 19.6%, 13.5%, and 13.0% of all manifestation occurrences, respectively, totaling about two-thirds. These are most prevalent because routing protocols are designed to ensure reachability: any logic error inevitably results in traffic being lost (Cut), misdirected (Blackhole), slowed (Delay), or isolated (Partition).

Finding 11: Security consequences caused by RPIBs primarily manifest as Cut, Delay, Blackhole, and Partition in networks, accounting for 20.8%, 19.6%, 13.5%, and 13.0%, respectively, totaling about two-thirds.

Implication: When monitoring security symptoms caused by RPIBs, Cut, Delay, Blackhole, and Partition are significant indicators of compromised networks.

Given that the four security symptoms discussed above (i.e., Cut, Delay, Blackhole, and Partition) represent the largest share of network damages, security teams should shift their primary focus toward these specific anomalies during network monitoring. By incorporating advanced detection techniques, e.g., [78], [79], [80], [81], into current monitoring infrastructures, we can identify these symptoms more accurately and in a timely manner.

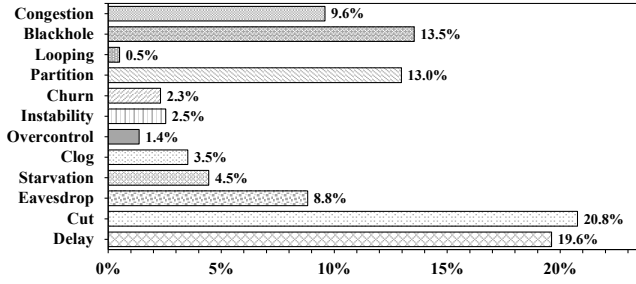


Fig. 16: Distribution of Network-Level Damages

C. Protocol Disparity Analysis.

This section compares BGP with IGP in terms of the security consequences (Fig. 17) and network-level manifestations (Fig. 18), examining whether differences in their design and application scenarios shape their security patterns. We compute the percentages of consequence or manifestation occurrences for BGP and IGP independently. Recall that BGP and IGP have a similar number of bugs in the dataset (see Table III). Thus, the distribution comparisons below also reflect the absolute bug counts.

Both protocols exhibit remarkably similar general consequences, with Disruption overwhelmingly dominating both BGP (84.6%) and IGP (87.8%). They also share the same top four most prevalent network-level manifestations: Cut, Delay, Blackhole, and Partition, which collectively account for 66% and 67.8% of BGP and IGP, respectively. This consistency stems from the fact that their core functional components for message processing, route computation, and neighbor maintenance are highly analogous.

Finding 12: Both the number and distribution of BGP and IGP are largely consistent across the 4 general security consequences and the 12 network-level manifestations, resulting in similar security implications.

Implication: Although BGP receives more attention in research, IGP bugs can indirectly lead to similar delivery and infrastructure damage; therefore, IGP deserves comparable security testing and research effort.

Existing bug discovery techniques are predominantly focused on BGP, e.g., [2], [57]. Because BGP and IGP operate on different mechanisms, these BGP-centric tools often cannot be applied to IGP. However, this finding shows that the security impact of IGP bugs is remarkably similar to that of BGP across all consequence classes. This indicates a significant gap in current network security assurance. Therefore, future research cannot neglect methodologies tailored to IGP, ensuring that intra-domain routing infrastructures receive the same level of rigorous security validation as BGP.

VII. RELATED WORK

In this work, we conduct a large-scale, systematic empirical study specifically targeting security bugs in routing protocol implementations, namely RPIBs. Existing empirical studies on network reliability span three dimensions: human factors,

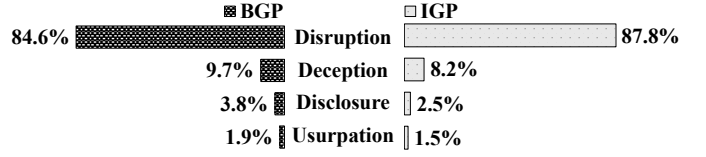


Fig. 17: General Security Consequences: BGP vs. IGP

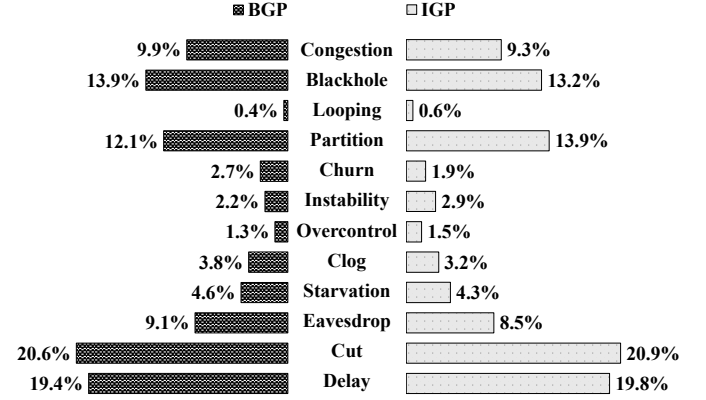


Fig. 18: Network-Level Manifestations: BGP vs. IGP

hardware, and software. This section distinguishes our work from these existing works.

Human Factors. Prior works [82], [83], [84], [85], [86] leverage configuration analysis, support forum mining, and operator surveys to demonstrate that misconfigurations and management complexity are predominant drivers of network outages, often demanding extensive manual effort to diagnose and resolve. Unlike the configuration-related software bugs studied in this paper, these prior studies examine how humans make mistakes when configuring a network.

Hardware. Studies on hardware failures [87], [88], [89], [90] examine device telemetry and connectivity events, revealing that hardware faults typically exhibit strong spatial correlation, often localized within specific devices. While hardware-centric research offers valuable perspectives on network reliability, they complement our software-centric focus on routing protocol implementations.

Software. From the software perspective, Yin et al. (2010) [53] provided a preliminary, coarse-grained characterization of bugs in routing implementations sixteen years ago. Our work distinguishes itself in three primary dimensions:

- **Relevance of target systems:** The protocol suites investigated by Yin et al., such as Quagga, have largely transitioned out of active maintenance. In contrast, we focus on modern, actively maintained artifacts, thus capturing current industry trends and the evolving security posture of contemporary routing software.
- **Dataset scale and openness:** Our study leverages a significantly larger dataset of 1,255 RPIBs, providing the statistical power necessary for more robust generalizations than previous small-scale studies. By contrast, the prior

study investigates a much smaller dataset of 364 RPIBs, which is also not publicly available.

- Granularity of analysis: Traditional taxonomies, including those used by Yin et al., often limit categorization to broad “memory” versus “semantic” distinctions. Our approach moves beyond these generalities by mapping bugs to routing-specific functional components. We specifically isolate “security-critical” RPIBs and analyze them through a multi-dimensional lens, identifying causes, triggers, and security impacts, to provide actionable, domain-specific insights for both developers and researchers.

To sum up, our work advances prior work by shifting from legacy systems to modern implementations that reflect current industry standards. By building an open dataset of 1,255 RPIBs, we significantly expand the empirical scope and provide a high-fidelity, routing-specific taxonomy. This enables granular mapping of root causes and security impacts to functional components, providing the targeted insights needed for actionable defenses.

VIII. CONCLUSION

In this paper, we present a systematic study of 1,255 real-world routing protocol implementation bugs (RPIBs) across three major routing suites, investigating their types, root causes, triggering conditions, and security impacts. From this study, we obtain twelve findings that are expected to provide developers and researchers with novel insights into routing system design and bug detection.

ACKNOWLEDGMENTS

We sincerely thank the anonymous reviewers for their suggestions. This work was partially supported by the National Natural Science Foundation of China (No. 62672233), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118), and the 111 Center (No. B26023).

ETHICS CONSIDERATIONS

This study exclusively analyzes known and publicly disclosed bugs. All data is sourced from established public repositories. Consequently, there is almost no risk of enabling immediate harm to end users or providing attackers with novel exploit vectors. Moreover, the use of public datasets eliminates concerns regarding non-disclosure agreements, proprietary code exposure, or developer privacy violations. We do not release functional exploit code or previously undisclosed attack techniques; our analysis is strictly historical and observational. Therefore, this study presents minimal ethical risk, and our focus remains on deriving generalizable insights to improve software security practices, without introducing new threats to the broader ecosystem.

REFERENCES

- [1] M. Prince, “Analysis of today’s CenturyLink/Level(3) outage,” The Cloudflare Blog, 2020. [Online]. Available: <https://blog.cloudflare.com/analysis-of-todays-centurylink-level-3-outage/>
- [2] D. dos Santos, S. Guiot, S. Dashevskiy, A. Amri, and O. Kerro, “Route to bugs: Analyzing the security of BGP message parsing,” Black Hat USA 2023, presentation slides, 2023. [Online]. Available: <https://i.blackhat.com/BH-US-23/Presentations/US-23-dosSantos-Route-to-Bugs-Analyzing-the-Security-of-BGP.pdf>
- [3] FRR Developers, “Frrouting project,” FRRouting project website, 2025. [Online]. Available: <https://frrouting.org/>
- [4] Q. Shi, X. Xiao, R. Wu, J. Zhou, G. Fan, and C. Zhang, “Pinpoint: fast and precise sparse value flow analysis for million lines of code,” in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. Association for Computing Machinery, 2018, p. 693–706. [Online]. Available: <https://doi.org/10.1145/3192366.3192418>
- [5] C. Y. Cho, V. D’Silva, and D. Song, “Blitz: Compositional bounded model checking for real-world programs,” in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 136–146. [Online]. Available: <https://doi.org/10.1109/ASE.2013.6693074>
- [6] Y. Xie and A. Aiken, “Scalable error detection using boolean satisfiability,” in *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL ’05. Association for Computing Machinery, 2005, p. 351–363. [Online]. Available: <https://doi.org/10.1145/1040305.1040334>
- [7] C. Wang, W. Wang, P. Yao, Q. Shi, J. Zhou, X. Xiao, and C. Zhang, “Anchor: Fast and precise value-flow analysis for containers via memory orientation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 1–39, 2023, art. no. 66. [Online]. Available: <https://doi.org/10.1145/3565800>
- [8] V.-T. Pham, M. Böhme, and A. Roychoudhury, “Aflnet: A greybox fuzzer for network protocols,” in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 460–465. [Online]. Available: <https://doi.org/10.1109/ICST46399.2020.00062>
- [9] J. Li, S. Li, G. Sun, T. Chen, and H. Yu, “Snpsfuzzer: A fast greybox fuzzer for stateful network protocols using snapshots,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2673–2687, 2022. [Online]. Available: <https://doi.org/10.1109/TIFS.2022.3192991>
- [10] J. Ba, M. Böhme, Z. Mirzamomen, and A. Roychoudhury, “Stateful greybox fuzzing,” in *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022, pp. 3255–3272. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity22/presentation/ba>
- [11] Q. Shi, X. Xu, and X. Zhang, “Extracting protocol format as state machine via controlled static loop analysis,” in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023, pp. 7019–7036. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity23/presentation/shi-qingkai>
- [12] Q. Shi, J. Shao, Y. Ye, M. Zheng, and X. Zhang, “Lifting network protocol implementation to precise format specification with security applications,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’23. Association for Computing Machinery, 2023, p. 1287–1301. [Online]. Available: <https://doi.org/10.1145/3576915.3616614>
- [13] V.-T. Pham, “Aflsmart++: Smarter greybox fuzzing,” in *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. IEEE, 2023, pp. 76–79. [Online]. Available: <https://doi.org/10.1109/SBFT59156.2023.00023>
- [14] J. Wang, B. Chen, L. Wei, and Y. Liu, “Superion: Grammar-aware greybox fuzzing,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 724–735. [Online]. Available: <https://doi.org/10.1109/ICSE.2019.00081>
- [15] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, “Directed greybox fuzzing,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’17. Association for Computing Machinery, 2017, p. 2329–2344. [Online]. Available: <https://doi.org/10.1145/3133956.3134020>
- [16] H. Huang, Y. Guo, Q. Shi, P. Yao, R. Wu, and C. Zhang, “Beacon: Directed grey-box fuzzing with provable path pruning,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 36–50. [Online]. Available: <https://doi.org/10.1109/SP46214.2022.9833751>
- [17] C. Luo, W. Meng, and P. Li, “Selectfuzz: Efficient directed fuzzing with selective path exploration,” in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2693–2707. [Online]. Available: <https://doi.org/10.1109/SP46215.2023.10179296>

- [18] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Network configuration synthesis with abstract topologies," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2017. Association for Computing Machinery, 2017, p. 437–451. [Online]. Available: <https://doi.org/10.1145/3062341.3062367>
- [19] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "NetComplete: Practical Network-Wide configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, 2018, pp. 579–594. [Online]. Available: <https://www.usenix.org/conference/nsdi18/presentation/el-hassany>
- [20] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. Association for Computing Machinery, 2016, p. 328–341. [Online]. Available: <https://doi.org/10.1145/2934872.2934909>
- [21] M. Zheng, Q. Shi, X. Liu, X. Xu, L. Yu, C. Liu, G. Wei, and X. Zhang, "Pardiff: Practical static differential analysis of network protocol parsers," *Proc. ACM Program. Lang.*, vol. 8, no. OOPSLA1, pp. 1208–1234, 2024. [Online]. Available: <https://doi.org/10.1145/3649854>
- [22] B. Shui, Y. Zhou, J. Wu, B. Xu, and Q. Shi, "Validating interior gateway routing protocols via equivalent topology synthesis," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. Association for Computing Machinery, 2025, p. 827–841. [Online]. Available: <https://doi.org/10.1145/3719027.3744826>
- [23] A. Cavalli, S. Maag, and E. M. de Oca, "A passive conformance testing approach for a manet routing protocol," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, ser. SAC '09. Association for Computing Machinery, 2009, p. 207–211. [Online]. Available: <https://doi.org/10.1145/1529282.1529326>
- [24] S. Bhosale and R. Joshi, "Conformance testing of ospf protocol," in *2008 IET International Conference on Wireless, Mobile and Multimedia Networks*. IEE, 2008, pp. 42–47. [Online]. Available: <https://doi.org/10.1049/cp:20080141>
- [25] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model-guided protocol fuzzing," in *Proceedings of the 2024 Network and Distributed System Security Symposium*. Internet Society, 2024. [Online]. Available: <https://dx.doi.org/10.14722/ndss.2024.24556>
- [26] M. Zheng, D. Xie, Q. Shi, C. Wang, and X. Zhang, "Validating network protocol parsers with traceable rfc document interpretation," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, pp. 1772–1794, 2025. [Online]. Available: <https://doi.org/10.1145/3728955>
- [27] Z. Luo, Q. Du, Y. Wang, A. Roychoudhury, and Y. Jiang, "Enhancing protocol fuzzing via diverse seed corpus generation," *IEEE Transactions on Software Engineering*, vol. 51, no. 9, pp. 2693–2709, 2025. [Online]. Available: <https://doi.org/10.1109/TSE.2025.3595396>
- [28] X. Song, L. Pei, J. Wu, Y. Zeng, G. He, C. Zuo, X. Liu, Q. Zhao, and S. Guo, "Protocolguard: Detecting protocol non-compliance bugs via llm-guided static analysis and dynamic verification," in *Proceedings of the 2026 Network and Distributed System Security Symposium*. Internet Society, 2026. [Online]. Available: <https://dx.doi.org/10.14722/ndss.2026.240521>
- [29] M. Zheng, C. Wang, X. Liu, J. Guo, S. Feng, and X. Zhang, "Rfcaudit: Ai agent for auditing protocol implementations against rfc specifications," in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2025, pp. 1221–1233. [Online]. Available: <https://doi.org/10.1109/ASE63991.2025.00105>
- [30] CZ.NIC Labs, "BIRD internet routing daemon," BIRD project website, 2025. [Online]. Available: <https://bird.network.cz/>
- [31] Holo Routing Contributors, "Holo project," Holo project repository on GitHub, 2025. [Online]. Available: <https://github.com/holo-routing/holo>
- [32] FRRouting Contributors, *FRRouting User Guide: Basic Setup*, 2026. [Online]. Available: <https://docs.frrouting.org/en/latest/setup.html>
- [33] P. Lumbis, "FRR: The most popular network router you've never heard of," The Next Platform, 2020. [Online]. Available: <https://www.nextplatform.com/connect/2020/10/26/frr-the-most-popular-network-router-youve-never-heard-of/1636827>
- [34] NVIDIA Corporation, *Cumulus Linux 5.16 Documentation: Layer 3 - FRRouting*, 2026. [Online]. Available: <https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-516/Layer-3/FRRouting/>
- [35] BIRD Developers, *BIRD Internet Routing Daemon: Introduction*, 2026. [Online]. Available: https://bird.network.cz/?get_doc&v=20&f=bird-1.html
- [36] P. Richter, G. Smaragdakis, A. Feldmann, N. Chatzis, J. Boettger, and W. Willinger, "Peering at peerings: On the role of ixp route servers," in *Proceedings of the 2014 ACM SIGCOMM Internet Measurement Conference*, ser. IMC '14. Association for Computing Machinery, 2014, p. 31–44. [Online]. Available: <https://doi.org/10.1145/2663716.2663757>
- [37] M. Prince, "Load Balancing without Load Balancers," Cloudflare Blog, 2013. [Online]. Available: <https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/>
- [38] J. Moy, "OSPF Version 2," RFC 2328, 1998. [Online]. Available: <https://www.rfc-editor.org/info/rfc2328>
- [39] R. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments," RFC 1195, 1990. [Online]. Available: <https://www.rfc-editor.org/info/rfc1195>
- [40] G. S. Malkin, "RIP Version 2," RFC 2453, 1998. [Online]. Available: <https://www.rfc-editor.org/info/rfc2453>
- [41] J. Chroboczek and D. Schinazi, "The Babel Routing Protocol," RFC 8966, 2021. [Online]. Available: <https://www.rfc-editor.org/info/rfc8966>
- [42] K. Lougheed, "Border Gateway Protocol (BGP)," RFC 1105, 1989. [Online]. Available: <https://www.rfc-editor.org/info/rfc1105>
- [43] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, 2006. [Online]. Available: <https://www.rfc-editor.org/info/rfc4271/>
- [44] S. N.C., S. Majumder, and T. Menzies, "Early life cycle software defect prediction. why? how?" in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 448–459. [Online]. Available: <https://doi.org/10.1109/ICSE43902.2021.00050>
- [45] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. Association for Computing Machinery, 2007, p. 529–540. [Online]. Available: <https://doi.org/10.1145/1315245.1315311>
- [46] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," in *Proceedings 2018 Network and Distributed System Security Symposium*, ser. NDSS 2018. Internet Society, 2018. [Online]. Available: <http://dx.doi.org/10.14722/ndss.2018.23158>
- [47] A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" in *15th USENIX Security Symposium (USENIX Security 06)*. USENIX Association, 2006, pp. 93–104. [Online]. Available: <https://www.usenix.org/conference/15th-usenix-security-symposium/milk-or-wine-does-software-security-improve-age>
- [48] A. Barbir, S. L. Murphy, and Y. Yang, "Generic Threats to Routing Protocols," RFC 4593, 2006. [Online]. Available: <https://www.rfc-editor.org/info/rfc4593>
- [49] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 392–401. [Online]. Available: <https://doi.org/10.1109/ICSE.2013.6606585>
- [50] H. Zhong and Z. Su, "An empirical study on real bug fixes," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 2015, pp. 913–923. [Online]. Available: <https://doi.org/10.1109/ICSE.2015.101>
- [51] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram, "How do fixes become bugs?" in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. Association for Computing Machinery, 2011, p. 26–36. [Online]. Available: <https://doi.org/10.1145/2025113.2025121>
- [52] B. Ray, D. Posnett, P. Devanbu, and V. Filkov, "A large-scale study of programming languages and code quality in github," *Commun. ACM*, vol. 60, no. 10, p. 91–100, 2017. [Online]. Available: <https://doi.org/10.1145/3126905>
- [53] Z. Yin, M. Caesar, and Y. Zhou, "Towards understanding bugs in open source router software," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 3, p. 34–40, 2010. [Online]. Available: <https://doi.org/10.1145/1823844.1823849>
- [54] Y. Li, X. Yin, Z. Wang, J. Yao, X. Shi, J. Wu, H. Zhang, and Q. Wang, "A survey on network verification and testing with formal methods: Approaches and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 940–969, 2019. [Online]. Available: <https://doi.org/10.1109/COMST.2018.2868050>

- [55] W. Sun, L. Xu, S. Elbaum, and D. Zhao, "Model-Agnostic and efficient exploration of numerical state space of Real-World TCP congestion control implementations," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, 2019, pp. 719–734. [Online]. Available: <https://www.usenix.org/conference/nsdi19/presentation/sun>
- [56] Z. Luo, J. Yu, F. Zuo, J. Liu, Y. Jiang, T. Chen, A. Roychoudhury, and J. Sun, "Bleem: Packet sequence oriented fuzzing for protocol implementations," in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023, pp. 4481–4498. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity23/presentation/luo-zhengxiong>
- [57] R. Singha, R. Mondal, R. Beckett, S. K. R. Kakarla, T. Millstein, and G. Varghese, "MESSI: Behavioral testing of BGP implementations," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 2024, pp. 1009–1023. [Online]. Available: <https://www.usenix.org/conference/nsdi24/presentation/singha>
- [58] D. Wang, P. Zhang, and A. Gember-Jacobson, "Expresso: Comprehensively reasoning about external routes using symbolic simulation," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. Association for Computing Machinery, 2024, p. 197–212. [Online]. Available: <https://doi.org/10.1145/3651890.3672220>
- [59] Y. Li, H. Zhang, J. Wang, Z. Wang, X. Yin, X. Shi, and J. Wu, "A general approach to generate test packets with network configurations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, p. 1362–1375, 2023. [Online]. Available: <https://doi.org/10.1109/TPDS.2023.3241433>
- [60] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, p. 11–33, 2004. [Online]. Available: <https://doi.org/10.1109/TDSC.2004.2>
- [61] R. Coltin, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," RFC 5340, 2008. [Online]. Available: <https://www.rfc-editor.org/info/rfc5340/>
- [62] K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "AddressSanitizer: A fast address sanity checker," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, 2012, pp. 309–318. [Online]. Available: <https://www.usenix.org/conference/atc12/technical-sessions/presentation/serebryany>
- [63] C. Crochet, J. Aoga, and A. Legay, "Formally discovering and reproducing network protocols vulnerabilities," in *Secure IT Systems: 29th Nordic Conference, NordSec 2024 Karlstad, Sweden, November 6–7, 2024 Proceedings*. Springer-Verlag, 2024, p. 424–443. [Online]. Available: https://doi.org/10.1007/978-3-031-79007-2_22
- [64] X. Feng, Q. Li, K. Sun, K. Xu, and J. Wu, "Exploiting cross-layer vulnerabilities: Off-path attacks on the tcp/ip protocol suite," *Commun. ACM*, vol. 68, no. 3, p. 48–59, 2025. [Online]. Available: <https://doi.org/10.1145/3689819>
- [65] M. A. Alim and T. G. Griffin, "On the interaction of multiple routing algorithms," in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '11. Association for Computing Machinery, 2011, pp. 1–12, art. no. 7. [Online]. Available: <https://doi.org/10.1145/2079296.2079303>
- [66] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "Crystalnet: Faithfully emulating large production networks," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. Association for Computing Machinery, 2017, p. 599–613. [Online]. Available: <https://doi.org/10.1145/3132747.3132759>
- [67] A. Abhashkumar, K. Subramanian, A. Andreyev, H. Kim, N. K. Salem, J. Yang, P. Lapukhov, A. Akella, and H. Zeng, "Running BGP in data centers at scale," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021, pp. 65–81. [Online]. Available: <https://www.usenix.org/conference/nsdi21/presentation/abhashkumar>
- [68] W. Deng, P.-D. Zhu, X.-C. Lu, and B. Plattner, "On evaluating bgp routing stress attack," *J. Commun.*, vol. 5, no. 1, pp. 13–22, 2010. [Online]. Available: <https://api.semanticscholar.org/CorpusID:16268076>
- [69] M. Schuchard, A. Mohaisen, D. Foo Kune, N. Hopper, Y. Kim, and E. Y. Vasserman, "Losing control of the internet: using the data plane to attack the control plane," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. Association for Computing Machinery, 2010, p. 726–728. [Online]. Available: <https://doi.org/10.1145/1866307.1866411>
- [70] C. Wen, Y. Liu, and S. Li, "A routing protocols fuzzing method based on man-in-the-middle," in *2022 2nd International Conference on Frontiers of Electronics, Information and Computation Technologies (ICFEICT)*. IEEE, 2022, pp. 491–496. [Online]. Available: <https://doi.org/10.1109/ICFEICT57213.2022.00092>
- [71] S. Gorbunov and A. Rosenbloom, "Autofuzz: Automated network protocol fuzzing framework," *IJCSNS*, vol. 10, no. 8, pp. 239–245, 2010. [Online]. Available: <https://people.csail.mit.edu/sergeyg/publications/autofuzz.pdf>
- [72] FRR Developers, "Frrouting user guide - protocols," FRRouting documentation, 2025. [Online]. Available: <https://docs.frrouting.org/en/stable-10.5/protocols.html>
- [73] CZ.NIC Labs, "BIRD configuration," BIRD documentation, 2025. [Online]. Available: https://bird.network.cz/?get_doc&v=30&f=bird-6.html
- [74] B. Al-Musawi, P. Branch, and G. Armitage, "Bgp anomaly detection techniques: A survey," *Commun. Surveys Tuts.*, vol. 19, no. 1, p. 377–396, 2017. [Online]. Available: <https://doi.org/10.1109/COMST.2016.2622240>
- [75] Y. Zhang and Z. M. Mao, "Effective diagnosis of routing disruptions from end systems," in *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*. USENIX Association, 2008, pp. 219–232. [Online]. Available: <https://www.usenix.org/conference/nsdi-08/effective-diagnosis-routing-disruptions-end-systems>
- [76] W. Li, Y. Li, and T. Chung, "ImPROV: Measurement and practical mitigation of collateral damage in RPKI route origin validation," in *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, 2025, pp. 3631–3647. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity25/presentation/li-weitong>
- [77] D. Mirdita, H. Schulmann, and M. Waidner, "SoK: An introspective analysis of RPKI security," in *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, 2025, pp. 3649–3665. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity25/presentation/mirdita>
- [78] A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé, "Analysis of country-wide internet outages caused by censorship," in *Proceedings of the 2011 ACM SIGCOMM Internet Measurement Conference*, ser. IMC '11. Association for Computing Machinery, 2011, p. 1–18. [Online]. Available: <https://doi.org/10.1145/2068816.2068818>
- [79] V. Giotsas, C. Dietzel, G. Smaragdakis, A. Feldmann, A. Berger, and E. Aben, "Detecting peering infrastructure outages in the wild," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. Association for Computing Machinery, 2017, p. 446–459. [Online]. Available: <https://doi.org/10.1145/3098822.3098855>
- [80] V. Giotsas, G. Smaragdakis, C. Dietzel, P. Richter, A. Feldmann, and A. Berger, "Inferring bgp blackholing activity in the internet," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. Association for Computing Machinery, 2017, p. 1–14. [Online]. Available: <https://doi.org/10.1145/3131365.3131379>
- [81] T. Holterbach, T. Alfroy, A. Phokeer, A. Dainotti, and C. Pelsner, "A system to detect Forged-Origin BGP hijacks," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 2024, pp. 1751–1770. [Online]. Available: <https://www.usenix.org/conference/nsdi24/presentation/holterbach>
- [82] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding bgp misconfiguration," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. Association for Computing Machinery, 2002, p. 3–16. [Online]. Available: <https://doi.org/10.1145/633025.633027>
- [83] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management," in *Proceedings of the 2009 USENIX Symposium on Networked System Design and Implementation*. USENIX Association, 2009, pp. 335–348. [Online]. Available: https://www.usenix.org/legacy/event/nsdi09/tech/full_papers/benson/benson_html/
- [84] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. Association for Computing Machinery, 2011, p. 159–172. [Online]. Available: <https://doi.org/10.1145/2043556.2043572>

- [85] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, "The cutting edge of ip router configuration," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, p. 21–26, 2004. [Online]. Available: <https://doi.org/10.1145/972374.972379>
- [86] K. Nagaraja, F. Oliveira, R. Bianchini, R. P. Martin, and T. D. Nguyen, "Understanding and dealing with operator mistakes in internet services," in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, ser. OSDI'04. USENIX Association, 2004, p. 5. [Online]. Available: https://www.usenix.org/legacy/events/osdi04/tech/full_papers/naragaraja/naragaraja_html/
- [87] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. Association for Computing Machinery, 2011, p. 350–361. [Online]. Available: <https://doi.org/10.1145/2018436.2018477>
- [88] E. B. Nightingale, J. R. Douceur, and V. Orgovan, "Cycles, cells and platters: an empirical analysis of hardware failures on a million consumer pcs," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. Association for Computing Machinery, 2011, p. 343–356. [Online]. Available: <https://doi.org/10.1145/1966445.1966477>
- [89] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 25–36. [Online]. Available: <https://doi.org/10.1109/DSN.2017.26>
- [90] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. Association for Computing Machinery, 2010, p. 193–204. [Online]. Available: <https://doi.org/10.1145/1807128.1807161>

APPENDIX A

MANIFESTATION OF SECURITY CONSEQUENCES

As discussed in §II-B and listed in Table II, the four general security consequences (Disclosure, Deception, Disruption, Usurpation) may manifest in 12 ways in a network. This section discusses their definitions and provides case studies.

There are 12 different manifestations, specifically, Congestion occurs when routing anomalies concentrate excessive traffic on certain links, causing the traffic demand to exceed link capacity. Blackhole occurs when large-volume traffic is drawn to a router that drops the packets rather than forwarding them toward the destination. Looping occurs when routers repeatedly forward traffic along a cyclic path, preventing it from ever reaching the destination. Partition occurs when some routers wrongly infer that they have been separated from the rest of the network, even though connectivity still exists. Churn occurs when routing paths change repeatedly and unnecessarily, causing continual fluctuations in forwarding behavior. Instability occurs when routers fail to converge on consistent routing results. Overcontrol occurs when excessive control messages, such as heartbeats, consume substantial link bandwidth and crowd out normal traffic. Clog occurs when large numbers of protocol messages exhaust device resources (e.g., CPU and memory). Starvation occurs when traffic is forwarded to a router that cannot deliver it, so the destination is never reached. Eavesdrop occurs when traffic is exposed to unintended observers, thereby creating opportunities for information leakage. Cut occurs when routers misperceive a reachable destination as unreachable, causing traffic to that destination to be abandoned. Lastly, Delay occurs when traffic

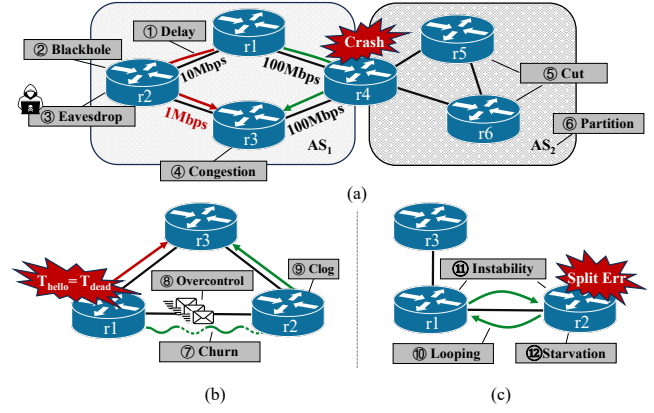


Fig. 19: Examples of the 12 network consequences: (a) a BGP bug (PR#14645) causing Delay, Blackhole, Eavesdrop, Congestion, Cut, and Partition; (b) an OSPF bug (PR#15014) causing Churn, Overcontrol, and Clog; and (c) an RIP bug (PR#18189) causing Looping, Instability, and Starvation.

is forwarded along a suboptimal path, increasing delivery latency relative to the intended route.

Case Study 1. Fig. 19 (a) illustrates a BGP network with six routers. Routers r_1 – r_4 are in AS_1 , whereas routers r_4 – r_6 are in AS_2 ; r_4 is the boundary router connecting the two ASes. In this scenario, a confidential message is sent from r_1 to r_3 . Two paths are available: $r_1 \rightarrow r_2 \rightarrow r_3$ (red path) and $r_1 \rightarrow r_4 \rightarrow r_3$ (green path). Because the green path has much higher bandwidth (100 Mbps vs. 1 Mbps), it is optimal and preferred.

Assume that an attacker can observe traffic at r_2 . To eavesdrop on the confidential message, the attacker exploits a BGP RPIB (PR#14645) with a malformed packet, crashes BGP on r_4 , and thereby forces traffic onto the red path through r_2 . Although the green path remains physically connected, r_1 considers it unavailable because BGP on r_4 is down. As a result, r_1 forwards the confidential message along the red path instead of the optimal green path.

This rerouting causes several consequences. The suboptimal path introduces Delay. Because the traffic passes through r_2 , the attacker can perform Eavesdrop, and r_2 become a Blackhole. The low-bandwidth link between r_2 and r_3 may also cause Congestion. Meanwhile, routers in AS_2 lose logical reachability to r_3 despite the physical link, resulting in Cut; they may also incorrectly infer that AS_1 is disconnected, causing Partition.

Case Study 2. Fig. 19(b) illustrates an OSPF network with three routers. There are two paths from r_1 to r_3 : the green path, $r_1 \rightarrow r_2 \rightarrow r_3$, and the red path, $r_1 \rightarrow r_3$. The green path is the optimal route.

In this scenario, the administrator misconfigures the link between r_1 and r_2 by setting the hello interval, which determines how often heartbeats are sent to detect whether r_2 is alive, equal to the dead interval, which specifies how long r_1 waits without receiving heartbeats before declaring r_2 dead. Due to an OSPF configuration bug (PR#15014), this invalid

TABLE VI: Keywords and their English variants.

Keyword	Variants
bug	bugs, bugged, bugging, buggy, debug, debugs, debugged, debugging, debugger, debuggers
defect	defects, defected, defecting, defective, defectively, defectiveness
error	errors, errored, erroring, erroneous, erroneously
fault	faults, faulted, faulting, faulty, faultier, faultiest, faultless, faultlessly
fix	fixes, fixed, fixing, fixer, fixers, bugfix, bugfixes, hotfix, hotfixes
issue	issues, issued, issuing
crash	crashes, crashed, crashing, crasher, crashers
security	secure, secures, secured, securing, securely, insecure, insecurity, insecurities
correct	correctly, incorrect, incorrectly
close	closes, closed, closing

configuration bypasses the configuration check. As a result, r_1 repeatedly oscillates between considering r_2 alive and dead. This behavior causes three consequences. First, r_1 repeatedly sends a large number of routing packets (e.g., information packets) in an attempt to maintain or reestablish the adjacency with r_2 , resulting in Overcontrol. Second, r_2 must continuously process these excessive hello packets, leading to Clog. Third, because r_1 repeatedly treats the r_1 - r_2 link as up and down, it keeps switching traffic between the green and red paths, causing Churn.

Case Study 3. Fig. 19(c) illustrates the consequences brought by an RIP bug (PR#18189). Router r_2 intends to send packets to r_3 via r_1 . However, due to a configuration bug, r_2 fails to enable split horizon, a mechanism designed to prevent forwarding loops. As a result, r_2 incorrectly advertises to r_1 that the next hop to r_3 is itself. Consequently, when r_2 forwards packets for r_3 to r_1 , r_1 sends them back to r_2 , causing the packets to circulate repeatedly between the two routers and forming a Looping. Although r_2 believes that a path to r_3 exists, the packets can never reach the destination, resulting in a Starvation. The root cause is the inconsistent routing results between r_1 and r_2 , which prevent correct convergence and lead to Instability.

APPENDIX B KEYWORDS

Table VI shows all keywords and their variants in data filtering.

APPENDIX C RPIB SUBTYPES

Table IV further divides each RPIB type into subtypes as per how they manifest. In Configuration bugs, *Parse Rejection* (T1) denotes valid configuration inputs being rejected; *Parse Corruption* (T2) denotes accepted inputs being incorrectly translated into internal configuration state; *Parse Validation Bypass* (T3) denotes invalid inputs bypassing validation; *Configuration Failure* (T4) denotes configuration changes

failing to take effect; *Configuration Corruption* (T5) denotes incorrect creation, update, or interpretation of configuration state; *Persistence Omission* (T6) denotes configuration data not being persisted; and *Persistence Malformation* (T7) denotes malformed persisted configuration data. Fig. 4(b) shows that *Configuration Corruption* dominates, accounting for 36.4% of configuration bugs, followed by *Configuration Failure* at 35.7% and *Parse Rejection* at 10.4%.

In Session bugs, *Establishment Failure* (T8) denotes failures in creating or negotiating a routing session, while *Unexpected Drop* (T9) denotes incorrect termination of an established session. Fig. 4(c) shows that *Establishment Failure* is slightly more common than *Unexpected Drop* (54.5% vs. 45.5%).

In (De)parsing bugs, *Parse Rejection* (T10) denotes valid protocol messages being rejected; *Parse Corruption* (T11) denotes protocol messages being incorrectly translated into internal representations; *Parse Validation Bypass* (T12) denotes invalid protocol messages bypassing validation; *Deparsing Failure* (T13) denotes failures in generating outgoing protocol messages; and *Deparsing Malformation* (T14) denotes malformed outgoing protocol messages. Fig. 4(d) shows that *Deparsing Malformation* is the largest subtype, accounting for 40.6%.

In Routing bugs, *Calculation Path Omission* (T15) denotes valid routes being omitted from route computation; *Calculation Logic Error* (T16) denotes incorrect route-computation results; *Storage Entry Omission* (T17) denotes computed routing entries not being installed or stored; and *Storage Entry Corruption* (T18) denotes stored routing entries being incorrect or corrupted. Fig. 4(e) shows that *Storage Entry Corruption* dominates, accounting for 59.6%.

In Synchronization bugs, *Propagation Failure* (T19) denotes failures in propagating routing information across protocols, while *Data Inconsistency* (T20) denotes inconsistent routing information after synchronization. Fig. 4(f) shows that *Propagation Failure* is more common than *Data Inconsistency* (60.5% vs. 39.5%).

In Logging bugs, *Content Error* (T21) denotes incorrect or misleading log contents, while *Generation Failure* (T22) denotes failures in producing expected logs. Fig. 4(g) shows that *Content Error* accounts for 45.9%, while *Generation Failure* accounts for 54.1%.

In System bugs, *System Management Failure* (T23) denotes failures that block normal process lifecycle operations, such as startup, shutdown, restart, or cleanup. *System Corruption* (T24) denotes abnormal runtime process states, such as crashes, zombie processes, or severe resource leakage. Fig. 4(h) shows that *System Management Failure* is much more common than *System Corruption* (72.2% vs. 27.8%).

Overall, developers and researchers can refer to this distribution of bug subtypes discussed above to benchmark the reliability of routing protocol implementations against this known distribution. Such benchmarking allows teams to quantify how their specific implementation stacks up against industry-wide failure patterns, transforming raw bug counts into a comparative metric for software maturity and operational readiness.