SIGCOMM '21 Posters and Demos: G: Non-interoperability Detection for Routing Protocol Implementations

Xi Jiang
The University of Chicago
xijiang9@uchicago.edu

Aaron Gember-Jacobson
Colgate University
agemberjacobson@colgate.edu

ABSTRACT

Network routing protocols help individual routers learn the network topology and select efficient routes, but the standards describing these protocols often contain ambiguous specifications. The abstract nature of the standards allows different implementations of the same routing protocol to have various interpretations of the specifications, causing them to experience non-interoperabilities when running in parallel. We present a technique for detecting such non-interoperabilities through specification mining for packet causal relationships.

CCS CONCEPTS

• **Networks → Protocol testing and verification**.

ACM Reference Format:
Xi Jiang and Aaron Gember-Jacobson. 2021. SIGCOMM '21 Posters and Demos: G: Non-interoperability Detection for Routing Protocol Implementations. In *SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21 Demos and Posters), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3472716.3472843

1 INTRODUCTION

Routing protocol standards define structures and algorithms for computing and disseminating routes. Some parts of a protocol standard are quite formal: e.g., packet structures are precisely defined in terms of field offsets, sizes, and value ranges. However, the majority of a standard is expressed in a natural language, which may be abstract or ambiguous. For example, OSPF's calculation of the shortest-path tree requires "a lookup in the [area's] link state database based on the [Link State ID]" [12], but the standard does not specify how to handle multiple entries with the same Link State ID [15]. Additionally, a standard may explicitly differentiate between absolute ("MUST"), ideal ("SHOULD"), and optional ("MAY") requirements [7]: e.g., the OSPF standard states that "originating summary-LSAs into stub areas [...] is optional" [12]. Consequently, different implementations of a routing protocol may embody different interpretations of the standard, leading to problems such as network instability, routing loops, or other routing anomalies when different implementations are used within/across routing domains.

Detecting interoperability issues arising from *flexibility* in a routing protocol standard (e.g., "MAY" requirements) is usually straightforward, because the standard typically indicates how an implementation should react if an optional feature is not supported. For example, the OSPF standard indicates that "a capability mismatch with a neighbor [...] will result in only a subset of the link state database being exchanged between the two neighbors" [12]. Examining routers' logs, internal states (e.g., using "show" commands on the routers), and/or exchanged packets (e.g., using tcpdump to examine options fields) is usually sufficient to detect that an implementation does not support an optional feature.

In contrast, detecting interoperability problems arising from *ambiguity* in a routing protocol standard is hard, because there is no definitive expected behavior. For example, as noted earlier, the OSPF standard does not specify how to handle multiple entries with the same Link State ID. Hence, some implementations may compute routes based on a falsified link state advertisement (LSA), while other implementations may chose the real LSA [15]. Ideally, a "de facto" or revised standard emerges—based on community norms or a major vendor's interpretation of the original standard—which allows conformance with the de facto/revised standard to be used as a means of detecting incompatibilities between protocol implementations. However, before a de facto/revised standard can emerge, we must first uncover the problematic differences in interpretation.

Prior studies have shown that numerous router software bugs cause interoperability issues among router vendors [19]. One example of a non-interoperability problem is the 2009 global internet slowdown[20]: Cisco routers could not correctly handled the long Autonomous System (AS) Path from MikroTik routers used by Supronet (a Czech Republic ISP), causing Cisco routers to experience repeated reboots and slowing down Internet traffic significantly.

Prior works have introduced three different approaches for detecting protocol interoperability issues: (1) analyze a model that embodies the protocol standard [17]—this is useful for finding inconsistencies or security vulnerabilities in the standard itself, but does not consider actual implementations; (2) compare an implementation of the protocol against a model that embodies the standard [6, 11, 13, 14, 18]—this is useful for determining whether an implementation deviates from the standard, but requires constructing a formal model that embodies the standard and does not elucidate differences between implementations; or (3) compare implementations of the protocol [16]—this can detect inconsistencies between implementations, but utilizes symbolic execution which requires access to implementations' source code.

We present a *black-box technique for detecting interoperability issues between routing protocol implementations based on the packets routers send and receive.* Crucially, we avoid the need to translate a protocol standard's natural language into a formal model; we only rely on the standard to determine the structure of packets, which, as noted above, is formally defined. Additionally, we do not require access to implementations' source code, which enables our technique to be applied to commercial protocol implementations.

2 RELATED WORK

In [9], the authors introduce two possible types of deviations in programmers' beliefs (as implied by their code) toward rules that systems must obey. First, given a fact that is known to be true ("MUST" beliefs), any deviation from the fact is considered a bug. Second, given that there is no known correct behavior, all possible behaviors ("MAY" beliefs) can be ranked by probability of occurrence and the most frequently occurring behavior can be viewed as a valid belief. This proposed system of deviation from beliefs can be applied to the study of routing protocols such that rules specified in RFCs should be treated as "MUST" beliefs that all protocol implementations must adhere to. Following the same approach, if there exists a specification that is open to interpretation, all possible interpretations of the specification across different implementations are "MAY" beliefs.

2.1 Verifying Conformance to "MUST" Beliefs

To check for conformance to "MUST" beliefs, researchers have developed various tools and techniques for comparing specific network protocol implementations to the standard documentation. [14] explores a C Model Checker (CMC) to model check the Adhoc On-demand Distance Vector (AODV)[8] networking protocol (implemented in C and C++). CMC uses an explicit state model checking to test as many system states as possible by varying the scheduling and execution of the processes. CMC defines a system state as the union of the states of all interacting concurrent agents (processes) as well as the shared memory content. As CMC checks the protocol implementation, it reports deadlock states, violations of protocol properties, and incorrect implementations of the protocol specifications.

As a continuation of the work, [13] concentrates on scaling CMC to more complex protocols, and it focuses on the Linux kernel's implementation of the Transmission Control Protocol (TCP)[5]. Instead of extracting and running a substantial amount of TCP implementation code from the Linux kernel as a closed system in the CMC context, the study runs the whole Linux kernel in CMC with threads that mimic the Linux TCP implementations. Additionally, the study largely mitigates the state explosion problem by storing only short signatures for explored states, processing only incremental differences between the states, and checking as many protocol behaviors as possible before running out of resources.

Aside from model checking, there are also other techniques that rely on random testing to verify protocol conformance to the standard. For example, [11] formalizes the QUIC[4] protocol specifications and uses randomized testers to verify the implementations' conformance to the formal specifications. These randomized testers create and verify interactions between different roles or communicating agents in the QUIC protocol, such as mimicking an interaction between a client and a server.

The above-mentioned techniques perform relatively well in verifying protocol adherence to the standard. However, many of these approaches rely on having a set of explicit-stated correct behaviors that the implementations must follow, which requires the related protocol specifications to be unambiguous. In the case of an ambiguous specification ("MAY" beliefs), it would be troublesome to define the correct behavior that the verifiers should anticipate from the implementations. Hence solely checking protocol implementations' compliance to the standard can only result in implementations that are as good and specific as the standard. As a result, there is a need to address these "MAY" beliefs due to standard ambiguities.

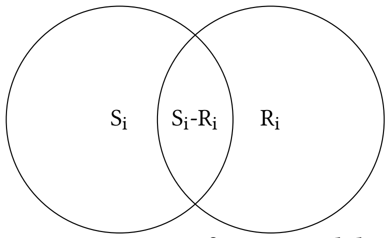


Figure 1: Concept of interoperability

2.2 Investigating the "MAY" beliefs

In this paper, we propose a new technique for detecting the *"MAY" beliefs-caused* non-interoperabilities among routing protocol implementations. There are several related studies that investigate the "MAY" beliefs by exploring the possibility of running multiple protocols (as well as different implementations) concurrently. The research conducted in [16] zooms in to explore non-interoperability among different implementations for the same protocol such as TCP, HTTP, SPDY, and SIP used in distributed system communications. The paper defines non-interoperability between two implementations as the difference (S_i-R_i) between the set of messages one implementation can send (S_i) and the set of messages the other implementation can receive (R_i) after the first i-1 messages have been sent and received, respectively (Figure 1).

Specifically, two implementations have non-interoperability if there exists a message that the sender can deliver to the receiver but the receiver will always reject due to non-compliance with its implementation. To search for non-interoperability between two implementations, the paper introduces a protocol interoperability checker (PIC). PIC finds non-interoperability by testing various inputs on the implementations and it adopts joint symbolic execution as well as guided search strategies to reduce path explosion by constraining the number of execution paths considered. However, the PIC proposed in this study is not specifically designed for discerning non-interoperabilities among routing protocol implementations and it is hard to extend the PIC to routing protocols, especially because routing protocol standards often do not have clear specifications for symbolic execution or modeling checking in general. At the same time, the PIC tackles the path explosion problem by limiting its search space which may lead to missing/unverified routing protocol behaviors. Our work aims to fill the gap by designing a detection method specifically for detecting routing protocol implementation non-interoperabilities using a black-box approach without symbolic execution or static analysis.

The finding of valid beliefs from a set of "may" beliefs in routing protocols and implementations can be used to enhance network performance and mitigate bugs. For example, [10] conducts an in-depth study to develop a bug-tolerant router (BTR) for eliminating bugs and misbehavior caused by routing protocols at run time. The BTR design runs several diversified instances of routing software (utilizing software and data diversity by introducing variations in codebase, version, update timing, connection, and protocol) under a router hypervisor layer. However, although the BTR exhibits overall improved robustness against routing protocol bugs, it is associated with the cost of additional delays in execution time which makes it less ideal as a long-term solution for overlooking the implementation non-interoperabilities.

3 APPROACH

Our technique for detecting routing protocol implementation non-interoperabilities relies on inferring and comparing the *packet causal relationships* for the selected implementations. After a routing protocol implementation sends (or receives) a packet *A*, there exists a sets of possible packets that the implementation expects to receive (or send) as compliant responses to *A*. We refer to the correlation between the sent (or received) packet and the set of expected responses as a packet causal relationship of the implementation. Our technique computes such relations of the implementations, allowing us to formalize the implementations' interpretation of the standard in terms of packet events. Moreover, inconsistencies in the relationships can serve as indicators of possible non-interoperabilities among the implementations: one implementation can forward a packet that it considers as a legitimate response, but the receiving implementation may always reject such a packet as it deems the packet as a non-compliant response.

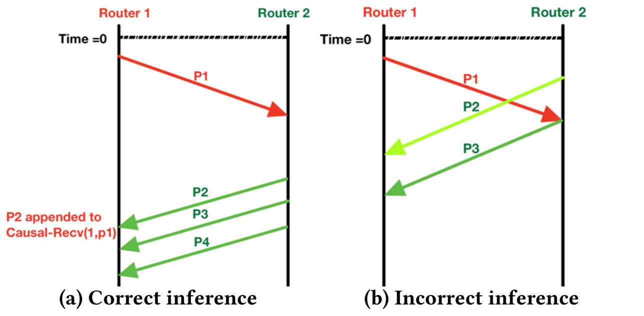
The main challenge in developing such a technique is to compute packet causal relationships that are both *accurate* and *extensive*. First, we want to ensure that the computed packet causal relationships are accurate, that is the reflected packets are indeed causally related. Second, we want to generate extensive packet causal relationships of the implementations, hence it is important to consider and analyze different networks scenarios.

Method overview. We create small-scale networks where each network runs a different implementation of the same routing protocol. Observing and analyzing the packets exchanged by the routers allows us to compute the packet causal relationships of the implementations. A naive approach to generate the packet causal relationships is as follows: After a packet *A* is sent (or received) by a router in the network, if packet *B* is the first packet received (or sent) by the same router, then we assume there is a causal relationship between the sending (or receiving) of *A* and the receiving (or sending) of *B*. More formally:

(a) After a packet *r* is received by a router running implementation *i*, the next packet sent by the router is appended to *Causal-Send*$_{i,r}$.

(b) After a packet *s* is sent by a router running implementation *i*, the next packet received by the router is appended to *Causal-Recv*$_{i,s}$.

For example, in Figure 2a, Router 1 sends packet P1 to Router 2 and subsequently receives three packets from Router 2. The first packet Router 1 received from Router 2 is P2, so P2 is appended to *Causal-Recv*$_{1,p1}$, but P3 and P4 are not.



(a) Correct inference (b) Incorrect inference

Figure 2: Naive causal relationship interference

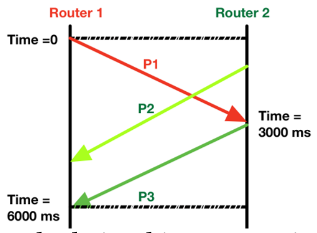


Figure 3: Causal relationship computation with delay

This process is applied to all routers within the same network, and we union the results to generate packet causal relationships for the specific implementation. As we compare the computed packet causal relationships of different implementations, we can flag disagreements across implementations as possible causes of non-interoperabilities.

However, as packets in networks are exchanged at a high frequency and the time spans between packet may be extremely small, there are often scenarios where a router receives multiple packets after sending a packet (or vice versa). This becomes especially troublesome as we are trying to determine which specific sent (or received) packet triggered the receiving (or sending) of another specific packet, which can result in inaccurate packet causal relationships. Consider the scenario in Figure 2b: after sending packet P1, Router 1 immediately receives packet P2 from Router 2. Router 1 thinks P2 is triggered by P1 and appends P2 to *Causal-Recv*$_{1,p1}$. But the actual packet that Router 2 used to respond to P1 is P3, while P2 is triggered by some other prior packet not relevant to P1. However, the naive approach fails to capture the true causal relationship.

Adding delay to improve accuracy. To exclude non-relevant packets from the computed packet causal relationships and achieve higher accuracy, we configure a fixed delay *TDelay* on all interfaces of the network. For example, a 3000ms *TDelay* implies that, when a router sends a packet, the receiving router will not receive and respond to the packet until after at least 3000ms. As a result, after a packet *A* is sent (or received) by a router in the network, instead of considering the first packet received (or sent) by the same router for the packet causal relationship, we look for the first packet received (or sent) by the same router after at least *2*TDelay* since packet *A* was sent (or received). *TDelay* should be set to a value that is greater than the variance in round trip time (RTT) and processing time (to ensure accurate causal analysis) and lower than the re-transmission timeout (to avoid spurious re-transmissions).

Figure 3 shows an example of the causal relationship computation with a *TDelay* of 3s applied to all interfaces. Router 1 sends a packet P1 to Router 2 at time 0 and we know P1 will not arrive at Router 2 until at least 3s later because of *TDelay*. And, if Router 2 decides to respond with a packet P3, we know P3 will not arrive at Router 1 until at least another 3s has passed. As a result, only P3 will be added to *Causal-Recv*$_{1,p1}$, because P3 is the first packet Router 1 received at least *2*TDelay* (6s) after sending P1. Adding the fixed delay allows us to exclude non-relevant packets such as P2, and thus helps to improve the accuracy of the computed packet causal relationships.

Using diverse network topologies to improve extensiveness. To further improve our technique, we adopt diverse network topologies when running routing protocols. As we introduce different network topologies, we alter network features such as the number of routers, interfaces, and neighbors. Specifically, we experimented with linear topologies with 2 or 5 routers and mesh topologies with 3 or 5 routers. In our experiments, we stopped seeing significant changes in the packet causal relationships after considering these four topologies, but additional topologies can be added to further improve completeness.

4 EVALUATION

Table 1: Packet causal relationships: general types

	FRR					BIRD				
	Snd(1)	Snd(2)	Snd(3)	Snd(4)	Snd(5)	Snd(1)	Snd(2)	Snd(3)	Snd(4)	Snd(5)
Rcv(1)	✓	✓	✓	✓	✓	✓	✓	∅	✓	✓
Rcv(2)	✓	✓	✓	∅	∅	✓	✓	✓	∅	∅
Rcv(3)	∅	∅	∅	✓	∅	∅	✓	✓	∅	∅
Rcv(4)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Rcv(5)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

(1) Hello, (2) DB Description, (3) LS Update, (4) LS Request, (5) LS Acknowledge

Table 2: Packet causal relationships: greater sequence number in LSA for LSU and LSAck

	FRR		BIRD	
	Snd(LSU)	Snd(LSAck)	Snd(LSU)	Snd(LSAck)
Rcv(LSU) with greater LS-SN in LSA	✓	✓	✓	✓
Rcv(LSAck) with greater LS-SN in LSA	∅	∅	✓	∅

To evaluate the effectiveness of the technique, we apply it to the FRRouting [2] and BIRD [1] implementations of OSPF. We run these implementations in Docker containers connected by virtual links. *TDelay* is introduced using the Pumba [3] chaos testing tool. We set *TDelay* to 900 ms, because the reduction in the unobserved packet causal relationships plateaued with these amount of delay.

Coarse-grained evaluation. Table 1 shows the computed packet causal relationships for packets differentiated by OSPF packet type, where missing relationships are represented with ∅values. We observe clear discrepancies between the two implementations which can be flagged as possible causes of non-interoperabilities. If we look at the packet causal relationships after sending an Link State Request (LSR) packet, we can see that FRRouting has one non-observed packet causal relationship and BIRD has two. Likewise, after sending an Link State Update (LSU) packet, FRRouting's only non-observed packet causal relation is on receiving an LSU packet while BIRD's only non-observed packet causal relation is on receiving a Hello packet.

Fine-grained evaluation. Table 2 demonstrates the more specific computed packet causal relationships: whether the sending of LSU or Link State Acknowledgment (LSAck) packets can trigger the receiving of LSU or LSAck packets with greater Link State Advertisement (LSA) sequence numbers (LS-SN). We can see that these two implementations have different interpretations regarding the expected behavior after the sending of LSU or LSAck packets: For example, after sending a LSU packet, FRRouting's causal relationship only observes the receiving of LSU packets with greater LS-SN while BIRD's causal relationship observes the receiving of both LSU and LSAck packets with greater LS-SN.

Such differences show that the two implementations of OSPF understand the protocol standard differently in terms of what packets to receive as responses to the sending of specific packets (For example, the sending of LSR and LSU packets in Table 1 and the sending of LSU packets in Table 2). These are indicators of possible non-interoperabilities of the two implementations which calls for verifying whether they can interoperate with each other even if they have these different understandings. Note that in both experiments, although the receive-send direction causal relationships are not shown, they are completely consistent with the send-receive direction causal relationships depicted in the tables.

5 CONCLUSION

By comparing the packet causal relationships in different implementations, we can formalize and capture the differences in the implementations' understandings of the standard. If the packet causal relationships are completely identical for the implementations, their behaviors in terms of packet processing should also be completely identical and there should be no non-interoperability. However, if there are differences in the implementations' packet causal relationships, although they do not guarantee non-interoperabilities, they can be indicators of possible non-interoperabilities.

To improve the accuracy and extensiveness of our packet causal relationships, we introduce fixed delays to the network interfaces and diverse network topologies during traffic capture. Our technique is proven to be effective as we detect packet causal relationship discrepancies when we apply the technique to the FRRouting and BIRD implementations of OSPF.

For future work, we want to validate our black-box inferences by examining the implementation source code. Furthermore, through means such as packet injection, we want to verify whether (or what fraction of) our flagged potential causes of non-interoperabilities indeed lead to bugs. We also aim to scale our system to consider more packet fields and other router features such as router states during the packet causal relationship computations.

REFERENCES

[1] [n.d.]. The BIRD Internet Routing Daemon Project. https://bird.network.cz.
[2] [n.d.]. FRRouting Protocols. https://frrouting.org.
[3] [n.d.]. Pumba. https://github.com/alexei-led/pumba/.
[4] [n.d.]. QUIC: A UDP-Based Multiplexed and Secure Transport draft-ietf-quic-transport-27. ([n. d.]). https://tools.ietf.org/html/draft-ietf-quic-transport-27.
[5] 1981. Transmission Control Protocol. RFC 793. https://doi.org/10.17487/RFC0793

[6] Silva Alexandra. 2021. Prognosis: Black-Box Analysis of Network Protocol Implementations. (2021).

[7] Scott O. Bradner. 1997. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119. https://doi.org/10.17487/RFC2119

[8] Samir R. Das, Charles E. Perkins, and Elizabeth M. Belding-Royer. 2003. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561. https://doi.org/10.17487/RFC3561

[9] Dawson Engler, David Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. 2001. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. *Symposium on Operating Systems Principles* 35 (09 2001).

[10] Eric Keller, Minlan Yu, Matthew Caesar, and Jennifer Rexford. 2009. Virtually Eliminating Router Bugs. In *CoNEXT*.

[11] Kenneth L. McMillan and Lenore D. Zuck. 2019. Formal specification and testing of QUIC. In *SIGCOMM*.

[12] John Moy. 1998. OSPF Version 2. RFC 2328. https://doi.org/10.17487/RFC2328

[13] Madanlal Musuvathi and Dawson R. Engler. 2004. Model checking large network protocol implementations. In *NSDI*.

[14] Madanlal Musuvathi, David Y. W. Park, Andy Chou, Dawson R. Engler, and David L. Dill. 2003. CMC: a pragmatic approach to model checking real code. *ACM SIGOPS Operating Systems Review* 36, SI (2003).

[15] Gabi Nakibly, Eitan Menahem, Ariel Waizel, and Yuval Elovici. 2013. Owning the Routing Table–Part II. In *BlackHat*.

[16] Luis Pedrosa, Ari Fogel, Nupur Kothari, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. Analyzing protocol implementations for interoperability. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[17] Adi Sosnovich, Orna Grumberg, and Gabi Nakibly. 2013. Finding Security Vulnerabilities in a Network Protocol Using Parameterized Systems. In *25th International Conference on Computer Aided Verification (CAV)*.

[18] Adi Sosnovich, Orna Grumberg, and Gabi Nakibly. 2017. Formal Black-Box Analysis of Routing Protocol Implementations. *CoRR* abs/1709.08096 (2017). arXiv:1709.08096 http://arxiv.org/abs/1709.08096

[19] Zuoning Yin, Matthew Caesar, and Yuanyuan Zhou. 2010. Towards understanding bugs in open source router software. *ACM SIGCOMM Computer Communication Review* 40, 3 (2010), 34–40.

[20] Earl Zmijewski. [n.d.]. Reckless Driving on the Internet. ([n. d.]). https://blogs.oracle.com/internetintelligence/reckless-driving-on-the-internet.