Non-interoperability Detection for Routing Protocol Implementations

Xi Jiang
Colgate University
xjiang@colgate.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. 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, 2 pages. https://doi.org/10.1145/3472716.3472843

1 MOTIVATION

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. 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.

Prior studies have shown that numerous router software bugs cause interoperability issues among router vendors [11]. One example of a non-interoperability problem is the 2009 global internet slowdown[12]: 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.

SIGCOMM '21 Demos and Posters, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8629-6/21/08.
https://doi.org/10.1145/3472716.3472843

Prior works have introduced three different approaches for detecting protocol interoperability issues: (1) analyze a model that embodies the protocol standard [9]—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 [4–7, 10]—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 [8]—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 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*. 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.

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 900ms *TDelay* implies that, when a router sends a packet, the receiving router will not receive and respond to the packet until after at least 900ms. 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).

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.

3 PRELIMINARY RESULTS & FUTURE WORK

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. Table 1 shows the computed packet causal relationships for packets differentiated by OSPF packet type, where missing relationships are represented with ∅values. Table 2 demonstrates the more specific computed packet causal relationships: whether the sending (or receiving) of Link State Update (LSU) or Link State Acknowledgment (LSAck) packets can trigger the sending (or receiving) of LSU or LSAck packets with greater Link State Advertisement (LSA) sequence numbers (LS-SN). 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.

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	∅	∅	✓	∅

We observe clear discrepancies between the two implementations which can be flagged as possible causes of non-interoperabilities. 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] Silva Alexandra. 2021. Prognosis: Black-Box Analysis of Network Protocol Implementations. (2021).
[5] Kenneth L. McMillan and Lenore D. Zuck. 2019. Formal specification and testing of QUIC. In *SIGCOMM*.
[6] Madanlal Musuvathi and Dawson R. Engler. 2004. Model checking large network protocol implementations. In *NSDI*.
[7] 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).
[8] 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)*.
[9] 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)*.
[10] 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
[11] 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.
[12] Earl Zmijewski. [n.d.]. Reckless Driving on the Internet. ([n. d.]). https://blogs.oracle.com/internetintelligence/reckless-driving-on-the-internet.