Causality Analysis in Control Plane Verification

Author(s):
Chen, Yu; Schneider, Tibor; Vanbever, Laurent

Publication date:
2023-12-08

Permanent link:
https://doi.org/10.3929/ethz-b-000643612

Originally published in:
https://doi.org/10.1145/3630202.3630237

Funding acknowledgement:
851809 - From Network Verification to Synthesis: Breaking New Ground in Network Automation (EC)

Causality Analysis in Control Plane Verification

Yu Chen
ETH Zürich
yuchen14@ethz.ch

Tibor Schneider
ETH Zürich
sctibor@ethz.ch

Laurent Vanbever
ETH Zürich
lvanbever@ethz.ch

ABSTRACT

Control plane verification promises to help operators build reliable networks by reporting a counterexample that violates the specification. However, a single counterexample imposes a major challenge for operators to understand and repair the violation.

To improve the usability of control plane verification, we present the first verifier computing the space of *all* specification violations as a symbolic expression. Our prototype implementation computes the *causality* between the network routing state and the external routing inputs that induce that state.

Describing the space of all violations helps operators address the root cause of the violation, while presenting the space as a symbolic expression allows operators to further manipulate the output to inspect certain aspects of the problem.

ACM Reference Format:
Yu Chen, Tibor Schneider, and Laurent Vanbever. 2023. Causality Analysis in Control Plane Verification. In *Proceedings of the CoNEXT Student Workshop 2023 (CoNEXT-SW '23), December 8, 2023, Paris, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3630202.3630237

1 INTRODUCTION

Over the past decade, the research community has proposed a large number of network verification systems. Control plane verification in particular promises to analyze the network-wide configuration and verify specifications against *all* external routing inputs, not just the ones that are currently received. The industry, however, has been hesitant in adopting such systems. One reason for this slow adoption is that these systems are challenging to use.

In case of a specification violation, control plane verifiers report a counterexample which consists of a violating network state and concrete external routing inputs that induce that state. Operators must repair their configuration based on this single counterexample. This is challenging for at least three reasons:

1. A single counterexample does not directly indicate bugs in the configuration. Operators can only guess which exact attributes cause the network to misbehave. Further, operators cannot identify whether a fix addresses the root cause of the violation.
2. Verifiers may return a corner case instead of a fundamental violation. The operator must fix that corner case before invoking the verifier again, hoping that the next counterexample is part of a more fundamental issue.
3. The counterexample may not describe a bug in the configuration, but it may point to an imprecise specification. For example, the counterexample can entail routing inputs the operator asserts to be impossible in practice without explicitly describing them in the specification. In this case, operators must refine the specification and rerun the verifier.

Instead of returning individual counterexamples, we propose a verification framework that computes an *exact* description of the *entire space* of external routing inputs that violate the specification. Instead of enumerating each counterexample, we describe the space using a single symbolic expression.

Such a symbolic expression addresses all three challenges mentioned above. First, having access to the entire violation space helps operators to identify the fundamental issue while ensuring that the repair addresses the entire violation space. Second, the symbolic expression describes every violation simultaneously, helping operators to identify corner cases and focus on major issues first. Third, the symbolic expression can be further refined and simplified, e.g., by removing the routing inputs that the operator asserts will never occur. Operators can also substitute symbols with concrete values and let automated tools simplify the resulting expression.

Generating such a symbolic expression requires a verifier to analyze the *causality* between the routing behavior and external routing inputs, i.e., how different routing inputs affect the network-wide routing state. Unfortunately, existing verification systems cannot express this causality due to limitations in their network encodings. Most systems rely on SMT solvers [1], which can only find individual counterexamples. Others use Binary Decision Diagrams (BDD) [2] to capture the local causality of a router interface, but they fail to analyze the network-wide routing as BDD cannot capture the route selection process efficiently.

We design a new control plane verification framework that captures the causality between the external routing inputs and the network routing states using symbolic expressions. The framework takes as input: *(i)* a network configuration; and *(ii)* a network specification. It first computes a mapping between a router's selected route and the routes of its neighbors based on each router's configuration. It then combines these mappings to compute a mapping between the external routing inputs and the network-wide routing state. In the end, it computes a symbolic expression describing the space of *all* external routing inputs that result in a network routing state that violates the specification.

Precisely describing all external routing inputs that violate a specification as a symbolic expression has its cost: it takes a lot of resources to generate and simplify the routing inputs, both in computation time and memory usage. In addition, when the network size increases, the expression can be more complex to understand than a single counterexample without further parsing. In §3, we show how time and memory usage grows exponentially with the network size. We discuss possible optimizations in §4.

CoNEXT-SW '23, December 8, 2023, Paris, France

ACM ISBN 979-8-4007-0452-9/23/12.
https://doi.org/10.1145/3630202.3630237

2 DESIGN

We compute a symbolic expression that describes all external routing inputs that violate the specification in three steps.

Step 1: Local causality. We first compute the *local causality*, a mapping from a router's *local environment* to its selected route. The local environment contains external routing inputs the router receives and the selected routes of its neighbors. The computation consists of two sub-steps. First, we transform the selected route of each neighbor by exploring all execution paths through both the neighbor's export policy and the router's import policy. Each execution path describes: *(i)* the received route; and *(ii)* the conditions when the route traverses the path.

Second, we symbolically perform the route selection process. For each execution path p, we compute the local environment for p's route by combining two conditions: *(i)* the route traverses p; and *(ii)* the route is preferred over all other received routes.

Step 2: Global causality. Based on the local causality, we compute the *global causality* between the external routing inputs and the global routing state, i.e., the selected routes of all routers in the network. To that end, we check whether each global routing state is reachable, i.e., whether there exist external routing inputs that result in the given state. To check a global routing state's reachability, we: *(i)* check that all local environments can be satisfied simultaneously; and *(ii)* ensure the absence of routing loops.

We then combine the local environments of all reachable states to remove the dependency on neighbors' selected routes. For each reachable state, we first compute its propagation graph, an acyclic-directed graph representing how routes are propagated in the network. We then repeatedly substitute the neighbor's selected routes in all local environments in an order that follows the propagation path in reverse. Finally, we combine all environments and yield a single symbolic expression describing the external routing inputs that lead to the given global routing state.

Step 3: Violation. In the final step, we generate a description of all external routing inputs that violate the specification. To that end, we collect all global routing states for which both the external routing inputs and the resulting state violate the specification. We then combine all collected routing inputs using a logical disjunction into a symbolic expression and perform boolean simplification.

3 PROTOTYPE

We implement a prototype with around 1k lines of Haskell code. We use the network in Fig. 1 to demonstrate our output. The network should advertise all prefixes it receives from its provider (`prov`) or peer to its customer (`cust`). The network fails to implement this specification: R1 does not advertise routes from the provider to the customer as the community `to-c` is missing. In the prototype, we encode the specification as Expr. (1), where p is a symbolic prefix.

$$prov.send(p) \vee peer.send(p) \implies cust.rcv(p) \tag{1}$$

The prototype outputs the following expression describing the entire space of external routing inputs that violate the specification:

$$\begin{gathered}\left(prov.send(p) \wedge \overline{peer.send(p)}\right) \vee P_c \\ P_c := p \in cust\text{-}pl \wedge cust.send(p)\end{gathered} \tag{2}$$

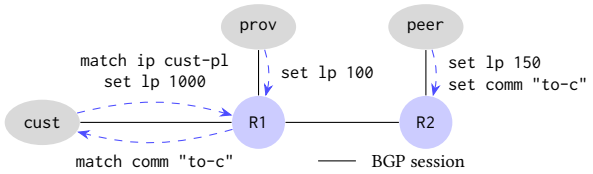

Figure 1: An example network

Expr. (2) shows that the specification is violated whenever only the provider advertises a prefix. This clearly identifies the fundamental violation: the provider's prefix is blocked from being advertised to the customer. In addition, the expression P_c describes a corner case where the customer does not receive the prefix advertised by itself, which points to an imprecise specification. Users can filter out this corner case by restricting $cust.send(p)$ to *false* in Expr. (2).

We further profile the end-to-end running time and memory usage for verifying networks of different sizes. We configure the routers to run BGP, connected using an iBGP route reflection topology with two route reflectors. We also configure well-known business relationships for three external neighbors (a provider, a peer, and a customer). The prototype takes 50ms for a network with 6 internal routers, while 200s for a network with 10 internal routers. The memory usage is 83MB and 852MB, respectively. The results indicate a scalability issue: the time and memory usage grows exponentially with the network size in Step 2.

4 OUTLOOK

We identify two dimensions to extend our prototype: its scalability and the generality of its specification language. We further describe new applications enabled by our framework.

Scalability. To improve the scalability of our prototype, we plan to adopt Multi-Terminal BDD to benefit from its compact encoding. In addition, we plan to include the specification already in Steps 1 and 2 (instead of only in Step 3) to simplify the global causality.

Specification Language. We intend to extend the specification language supported by our prototype by allowing operators to express high-level intent on both the routing state *and* the forwarding state of the network.

Applications. A description of the space of all external routing inputs that violate a specification can enable new applications for improving network operations. In particular, we envision a system that attaches probabilities to the space of external routing inputs. This allows operators to explore the space of violations by ordering them according to their likelihood.

ACKNOWLEDGMENTS

The research leading to these results was supported by an ERC Starting Grant (SyNET) 851809.

REFERENCES

[1] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. 2017. A general approach to network configuration verification. In *SIGCOMM*.
[2] S. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese. 2016. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *OSDI*.