Answering Why-Not Queries in Software-Defined Networks with Negative Provenance

Yang Wu[⋆∘§] Andreas Haeberlen[⋆] Wenchao Zhou[†] Boon Thau Loo[⋆]

[⋆]University of Pennsylvania [†]Georgetown University [§]Student [∘]Presenter

Finding problems in complex networks has always been challenging, as the substantial literature on network debugging and root-cause analysis tools can attest. However, the advent of software-defined networks (SDN) has added a new dimension to the problem: networks can now be controlled by programs, and, like all other programs, these programs can have bugs.

Existing network debuggers, such as `ndb` [3] or SNP [4], approach this problem by offering a kind of backtrace, analogous to a stack trace in a conventional debugger, that can link an observed effect of a bug to its root causes. For instance, suppose the administrator notices that a server is receiving requests that should have been handled by another server. The administrator can then trace the requests to the last-hop switch, where she might find a faulty flow entry; she can trace the faulty flow entry to a statement in the controller program that was triggered by a certain condition; and she can continue this process recursively until she reaches a set of root causes. The result is the desired backtrace: a causal chain of events that explains how the observed event came to pass. We refer to this as the positive provenance [1] of the event.

Positive provenance is great for debugging complex interactions, but there are cases that it cannot handle. For instance, suppose that the administrator observes that a certain server is no longer receiving any requests of a particular type. The key difference to the earlier scenario is that the observed symptom is not a positive event, such as the arrival of a packet, that could serve as a lead and point the administrator towards the root cause. Rather, the observed symptom is a negative event: the absence of packets of a certain type. Negative events can be difficult to debug: provenance does not help, and even a manual investigation can be difficult if the administrator does not know where the missing packets would normally come from, or how they would be generated.

Our approach: We have been developing a solution to this problem that relies on *negative provenance* [2]. Our solution can be used to answer "Why not?" questions in SDNs, i.e., explain the absence of events.

Our technique generates negative provenance based on counterfactual reasoning, i.e., it examines the possible causes that could have produced the missing effect. For instance, it might be the case that the missing packets could only have reached the server through one of two upstream switches, and that one of them is missing a flow entry that would match the packets. Based on the controller program, we might then establish that the missing flow entry could only have been installed if a certain condition had been satisfied, and so on, until we either reach a positive event that can be traced with positive provenance, or a negative root cause (such as a missing entry in a configuration file).

However, generating negative provenance is substantially more challenging than positive provenance, because: 1) Unlike positive provenance, which can be conceptually viewed as a subgraph of a finite graph that captures the dependencies among all system states and state changes, negative provenance does not have a well-defined model to start with there is an infinite set of negative events! 2) Whenever an effect can occur in more than one possible ways, positive provenance can simply report the specific cause, whereas negative provenance must consider each possible cause and show for each of them why it did not occur. 3) The overwhelming complexity of a navely-presented negative provenance might limit its usability.

The key insight behind our approach is to rule out most of the possible explanations as inconsistent with the controller program, which can only produce a few different flow entries, under very specific conditions. Further, we use several heuristics to produce a more compact representation of the (often verbose) negative provenance.

We have implemented an early prototype of a negative provenance engine for SDNs. Our initial experience confirms that, with careful program analysis and with appropriate summarization and pruning, negative provenance can answer concrete "Why not?" questions and produce surprisingly simple, high-quality explanations. We are currently working towards a more efficient and practical network debugger.

References

[1] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proc. ICDT*, 2001.

[2] A. Chapman and H. V. Jagadish. Why not? In *Proc. SIGMOD*, 2009.

[3] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. Where is the debugger for my software-defined network? In *Proc. HotSDN*, 2012.

[4] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure network provenance. In *Proc. SOSP*, Oct. 2011.