Integrating Verification and Repair into the Control Plane

Aaron Gember-Jacobson
Colgate University
agemberjacobson@colgate.edu

Costin Raiciu
Univ. Politehnica of Bucharest
costin.raiciu@cs.pub.ro

Laurent Vanbever
ETH Zürich
lvanbever@ethz.ch

ABSTRACT

Network verification has made great progress recently, yet existing solutions are limited in their ability to handle specific protocols or implementation quirks or to diagnose and repair the cause of policy violations. In this positioning paper, we examine whether we can achieve the best of both worlds: full coverage of control plane protocols and decision processes combined with the ability to diagnose and repair the cause of violations. To this end, we leverage the happens-before relationships that exist between control plane I/Os (e.g., route advertisements and forwarding updates). These relationships allow us to identify when it is safe to employ a data plane verifier and track the root-cause of problematic forwarding updates. We show how we can capture errors before they are installed, automatically trace down the source of the error and roll-back the updates whenever possible.

1 INTRODUCTION

"Network outages are the new natural disasters"

—G. Clay Whittaker

Ensuring a network works according to operator policies is of crucial importance to airlines, content providers, banks and really all other enterprise networks worldwide. When such networks fail or behave incorrectly they cause massive disruption, ranging from grounded airplanes with stranded passengers to millions in lost revenue [37].

Using formal methods to synthesize a correct network directly from policies [5,14,33,35] can avoid problems from the start. Unfortunately, such methods require an iterative specification and refinement process that is lengthy and requires significant formal methods expertise, making it a non-starter for most networks. Instead, most network operators continue to make manual configuration changes during maintenance windows and use simple tests to detect problems.

Verifying the correctness of a configuration change is incredibly complex, because network control planes routinely involve millions of lines of code from different vendors [19] running in an unpredictable distributed environment. To manage this complexity, researchers have developed verification techniques that verify *a model of the control plane*. The models aim to capture the behavior and interactions of distributed protocols and range from simple graph abstractions [15, 18] to more complex models expressed in first-order logic [4, 16]. The models can be used not only to detect which subsets of traffic are improperly forwarded, but also to identify [16] and repair [17] the control plane elements (e.g., routing protocol instances or filters) that contributed to the policy violations. However, to make verification and repair tractable, the models often consider a fraction of the control plane's functionalities, ignore some of the "ugly" implementation details, and overlook implementation quirks specific to each vendor. Because of these discrepancies, properties holding on the model may not hold in practice, and vice-versa. A clear parallel exists in the security world where published safety proofs of protocols such as TLS 1.2 [11, 25] actually missed *many* attacks (e.g., Heartbleed [12]), because of the gap between the simplified model and the implemented protocol.

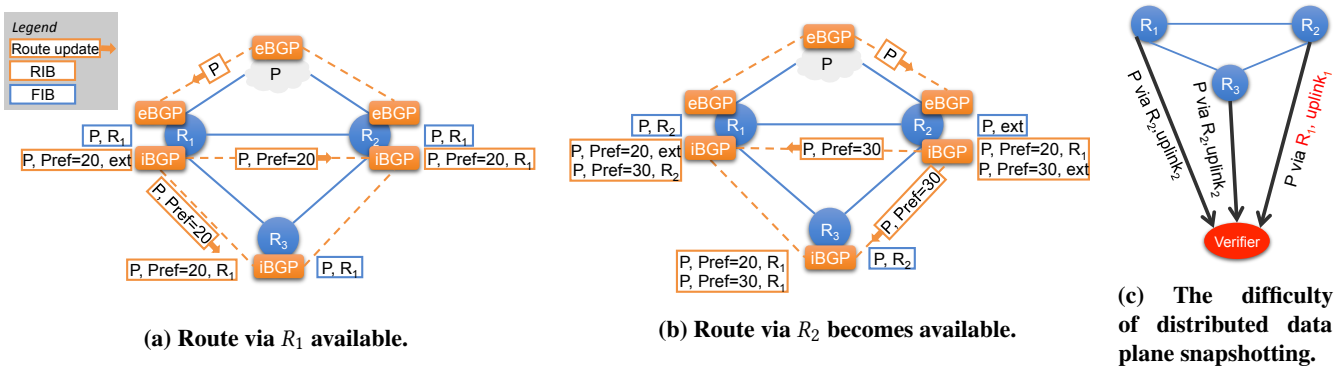
A second class of verification tools sidestep the complexity of the control plane by instead verifying *the control plane's output (i.e., the data plane)* [22, 24, 34]. The data plane implicitly captures all control plane functionality and is comparatively simpler to verify due to its limited operations (essentially, forwarding). However, without visibility into the control plane, data plane verifiers are unable to explain *why* a policy violation occurred. Consequently, the only immediate solution is to block data plane updates while network operators manually diagnose and correct the control plane. Blocking these updates can cause inconsistencies between the data and control planes that lead to further policy violations.

In this paper, we explore whether we can achieve the best of both worlds: full coverage of control plane protocols and decision processes combined with the ability to diagnose and repair the cause of violations. To this end, we propose an approach that *integrates verification and repair techniques into the operation of distributed control planes*. At a high level, our proposal is for each router to capture all control plane inputs and outputs, send them to a centralized data plane verifier, and only allow the data plane to be updated if the inputs and outputs are deemed correct.

Translating this simple idea to reality is tougher than it appears at first sight: the control plane is a distributed system and even obtaining a consistent snapshot of the control

HotNets-XVI, Palo Alto, CA, USA

DOI: 10.1145/3152434.3152439



(a) Route via R_1 available.

(b) Route via R_2 becomes available.

(c) The difficulty of distributed data plane snapshotting.

Figure 1: Example network to show the difficulty of network verification.

plane's inputs and outputs for verification is tricky; furthermore, selectively blocking data plane updates will result in incorrectly functioning protocols. To this end, our approach tracks happens-before relationships (HBRs) [26] between control plane inputs and outputs and uses the HBRs to support correct snapshotting, root-cause analysis and roll-back.

2 MOTIVATION

We use the network in Fig. 1 to show the complexity of maintaining policy compliance. Routers R_1, R_2, and R_3 are in the same administrative domain and use iBGP to disseminate external routes internally. There are two uplinks, via R_1 and R_2. Both R_1 and R_2 learn a route to prefix P via their respective eBGP sessions. The network policy states: R_2 is the preferred exit point when its uplink is up; otherwise, R_1 should be used. To implement the policy, operators configure a local preference (LP) of 30 on R_2 and 20 on R_1.

Verification amidst routing updates. Consider a scenario where only the route via R_1 is available (Fig. 1a), and then R_2 receives an advertisement for P on its uplink (Fig. 1b). In a few steps, all the routers will converge to a correct data plane in which both R_1 and R_3 forward traffic via R_2, as intended.

Unfortunately, analyzing this scenario using existing network verifiers is tricky. Some control plane verifiers [18] lack support for all of the routing protocols used in this network. Other control plane verifiers [4, 15, 16, 36] model all protocols and path selection criteria used in this network, but ignore vendor-specific implementation details that may apply in other scenarios—e.g., differences in BGP path selection rules across vendors [9, 21].

Feature coverage is not a problem for data plane verifiers [22,23,24,30,31,34], because they operate on the output of the actual control plane. However, they rely on a centralized snapshot of the data plane, which is difficult to construct, because routers may provide a snapshot of their forwarding information base (FIB) at slightly different times.[1] This is the case in Fig. 1c, where the FIB update at R_2 is just missed by the verifier (who gets a stale FIB entry), while R_1 and R_3 report their updated FIBs. Consequently, the data plane verifier will find a loop between R_2 and R_1 that sinks all traffic destined to P. This loop does not appear in practice, though, because of the way BGP works: R_1 and R_3 will only receive the update after R_2 installs it in its FIB.

Solving this problem is not easy. Verifying the actual control plane, as opposed to a model, is intractable. Waiting for the network to "converge" in order to capture a consistent data plane snapshot is elusive, as the network is constantly receiving new updates and can therefore be in a continuous state of change.

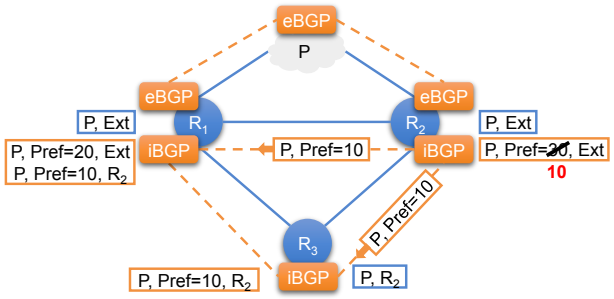
Correcting or preventing policy violations. As another example, Fig. 2a shows an ill-considered configuration update on R_2 that sets the LP to 10, which is lower than the value used by R_1. This will cause all routers to switch to using the uplink from R_1 instead of R_2, thus violating the policy. Assuming data or control plane verifiers can detect the violation, we still face the challenge of correcting or preventing the problem.

Data plane verifiers only analyze the control plane's output, and thus have no knowledge of how or why the control plane produced the problematic forwarding updates. Consequently, the only possible recourse is to block or revert the updates.[2] However, this creates an inconsistency between the data and control planes that may lead to further policy violations. For example, assume we block the FIB updates shown in Fig. 2 in order to preserve the behavior that traffic for P is sent via R_2 when its uplink is available. Now, assume R_2's uplink fails, and R_2 withdraws the route for P. The routers will not update their FIBs, because the control plane thinks the FIBs have the entries in Fig. 2b, which send traffic via R_1. However, due to the blocked FIB updates, the FIBs actually have the entries in Fig. 1b, so traffic for P will be sent to R_2 and dropped.

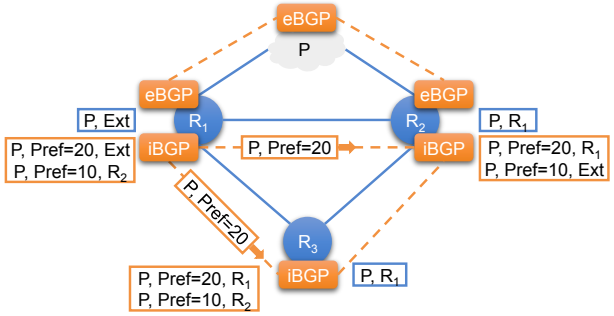
In contrast, tools that operate on the control plane can, in theory, identify [16] and correct [17] the root cause of the violation. However, current control plane repair tools [17] are limited in the protocols and policies they support and cannot correct the problem shown in Fig. 2. Consequently, operators must manually identify and implement a fix. This can be a

[1] This issue does not exist in software-defined networks (SDNs), because all changes are initiated by a central controller that can be co-located with the verifier.

[2] The former requires modifying a router's control plane to buffer forwarding updates until they have been verified, which is similar to interposing on the controller's output in SDNs [22, 24].



(a) Local pref configuration changed on R2: update propagates and makes R_1 switch to using its own uplink, violating the policy.



(b) R_1 announces own uplink route and this results in data plane policy violations at R_2 and R_3.

Figure 2: Ensuring policy compliance is hard, even in a simple BGP network.

lengthy process (on the order of hours) due to the complexity of analyzing how a change will impact the network [6].

3 INTEGRATING VERIFICATION AND REPAIR INTO THE CONTROL PLANE

The aforementioned issues stem from a disconnect between the operation of the actual control plane and the systems used for verification and repair: control plane verification [4, 15, 16, 18, 36] and repair [17] systems overlook (many) aspects of the actual control plane, while data plane verifiers [22, 23, 24, 30, 31, 34] only consider the control plane's output.

Addressing these issues requires *integrating verification and repair techniques into the operation of the control plane* (Fig. 3). Verification must be based on the *actual* decisions made by control plane, and corrections must be made *in the control plane* to address (impending) policy violations. This requires mechanisms to track and manipulate the control plane's actions.

We obtain such visibility and control by interposing on the control plane's inputs and outputs (I/Os) (§4). By monitoring the control plane's I/Os, we can build a consistent snapshot of the network's data plane to soundly verify policy compliance (§5). By computing the provenance of problematic control plane I/Os, we can identify which I/Os to actively block in order to maintain policy compliance (§6).

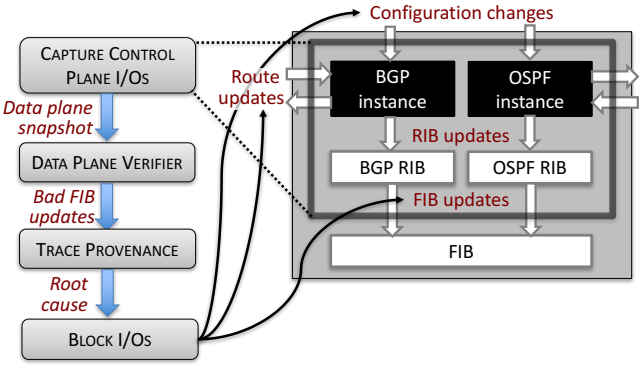


Figure 3: Integration of verification and repair mechanisms into the control plane

4 TRACKING THE CONTROL PLANE

To verify policy compliance and uncover the root cause of policy violations, we need to reason about the *origin* of any FIB entry and the way related control plane events propagate through the network. Prior works on network provenance [8, 16, 38, 40, 41] have used datalog-based models of the control plane [16, 28, 29] to enable such reasoning. However, as discussed in §2, producing high fidelity models of control plane algorithms is difficult. Fortunately, detailed modeling of a router's algorithms is unnecessary for identifying the cause(s) of FIB updates. We claim that *observing the inputs and outputs (I/Os) of a router's control plane and tracking the dependencies between them* provides sufficient information to safely verify and remediate problematic FIB updates.

4.1 Happens-Before Relationships

A router's control plane receives three types of input: protocol configurations, hardware status changes (e.g, link down), and route advertisements and withdrawals. Based on this input, protocol- and vendor-specific algorithms produce three main types of output: FIB entries, routing information base (RIB) entries, and route advertisements and withdrawals (for other routers). These outputs may be produced whenever a router receives new inputs. For example, when R_2 in Fig. 1 receives an eBGP advertisement for prefix P, it installs an entry for P in its BGP RIB.

Naturally, the input(s) on which an output depends must be received before the output is produced. For example, R_2 must receive the advertisement for P before it installs a RIB entry for P. In other words, there exists a *happens-before relationship* (HBR) [26] between the input and output. We can generically express this relationship as [router R receive protocol C advertisement for P] $\rightarrow$ [R install P in the C RIB], where [A] $\rightarrow$ [B] denotes I/O A happens before I/O B.

HBRs may also exist between two of a router's outputs or between one router's output and another router's input. For example, R_2 must install a RIB entry for P before it can install a FIB entry for P, and R_2 must send an iBGP advertisement for P before R_1 or R_3 can receive an iBGP advertisement for P. Written generically: [R install P in the C RIB] $\rightarrow$ [R

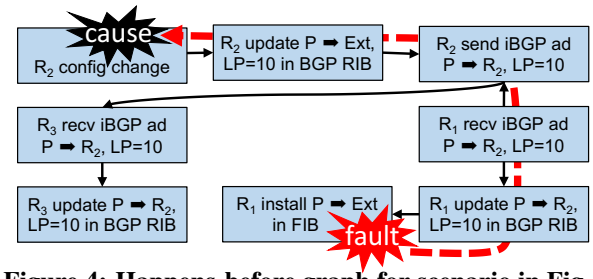


Figure 4: Happens-before graph for scenario in Fig. 2

install P in the FIB] and [R' send protocol C advertisement for P] $\rightarrow$ [R receive C advertisement for P][3].

The aforementioned HBRs apply to all common distributed routing protocols (e.g., BGP, OSPF, RIP, EIGRP). Additional HBRs apply to specific routing protocols, route redistribution and selection mechanisms, and network events (i.e., configuration or hardware status changes). For example, with BGP [R install P in BGP RIB] $\rightarrow$ [R send BGP advertisement for P], whereas with EIGRP [R install P in FIB] $\rightarrow$ [R send EIGRP advertisement for P].

4.2 Tracking HBRs

To determine a FIB entry's provenance, we must track control plane I/Os and HBRs in the live network. Capturing control plane I/Os is straightforward: most commercial router platforms provide a mechanism for logging control plane I/Os locally or to a remote server [10, 20], and open-source platforms [3] could be readily extended to provide such functionality. Identifying HBRs between the captured I/Os is more challenging: without access to a router's internals, we cannot track the data or control flow between a control plane's input and output functions. Instead, we must rely on properties of the I/Os themselves to infer their relationships. We intend to investigate several possible techniques:

Prefixes. All control plane outputs and incoming route advertisements contain a prefix. I/Os with the same prefix are frequently related: e.g., in Fig. 1, the route advertisement received by R_2, the RIB entry installed at R_2, and the route advertisements sent by R_2 and received by R_1 and R_3 are related. In contrast, the FIB entry for P installed on R_2 does not depend on the iBGP advertisements for P sent by R_2, despite their common prefix. Prefixes, like timestamps, can only be used to filter I/Os for possible HBRs.

Timestamps. The (wall-clock) time at which an I/O was captured can be used to identify I/Os that happened before other I/Os on the same router. However, I/Os that occur sequentially in time are not necessarily dependent: e.g., R_2 in Fig. 1 may install a FIB entry for P before sending iBGP advertisements for P, but the latter does not depend on the former. Thus, timestamps can be used to filter the HBRs considered/generated by other strategies, but timestamps cannot be used as the sole mechanism for identifying HBRs.

[3] R and R' may be connected at layers 1 (physical), 2 (link), or 3 (network), depending on the capabilities of the protocol.

Rule matching. In §4.1 we listed several "rules" that standard protocols follow. Given an I/O that matches the right-hand-side of a rule, we can search the (timestamp- and prefix-filtered) stream of I/Os for an I/O that matches the left-hand-size of the rule. A successful match implies there exists an HBR between the two I/Os. The disadvantage of this approach is that it requires understanding protocol standards to construct the rule set.

Pattern matching. To avoid the need for a detailed understanding of protocol implementations, we could instead look for I/O patterns in policy-compliant networks. If one I/O frequently occurs after another I/O, then we could infer the former must happen-before the latter. By looking for the same patterns in a broken network, we can produce a list of observed HBRs. This approach has the benefit of being fully automated, but we risk missing an important HBR, because it may not occur in the network(s) from which we derive patterns. Imperfect HBRs can lead to false positives and negatives during verification (§5). We intend to deal with this problem by adapting the behavior of our system according to a statistical confidence attached to each inferred HBR, only alerting and acting on a violation when it is high enough.

In practice, we expect a combination of these (and other) techniques will be necessary to obtain suitable accuracy.

4.3 Happens-before Graph

The HBRs that exist between control plane I/Os are often quite complex. In particular, an I/O may both a consequent and an antecedent of other I/Os: e.g., the RIB entry for P on R_2 in Fig. 1 requires the eBGP advertisement for P and is required for the FIB entry and iBGP advertisements for P.

To track these complex relationships, we can aggregate the observed HBRs into a *happens-before graph* (HBG). Vertices correspond to specific control plane I/Os, and directed edges represent HBRs. Fig. 4 depicts the HBG for the scenario in Fig. 2. In the next two sections, we discuss how the HBG helps us safely verify policy compliance during distributed routing updates (§5) and address the root cause of policy violations while keeping the control and data plane in sync (§6). Note that SDNRacer [13, 32] also relies on HBGs to verify correctness, but SDNRacer is limited to centralized control-planes, can only detect one type of correctness issue (race conditions), and cannot help in identifying root causes.

5 DETECTING POLICY VIOLATIONS

As discussed in §2, data plane verifiers require a consistent global view of the data plane to avoid missing violations or raising false alarms. In particular, they require a snapshot that reflects the FIB entries a packet would encounter as it traverses the network at a specific instance in time.[4] This ensures the verifier detects all transient *and* persistent violations

[4] In reality, packets take time to traverse the network and encounter router's FIBs at different instances in time. Thus, a lack of violations across consecutive consistent data plane snapshots does not strictly guarantee a packet does not violate a policy [39]. However, HBGs could be used to construct

for properties that should always be true (e.g. traffic should never bypass a firewall).

Such a snapshot can be easily computed given an exact (i.e., total) order of FIB updates. However, FIB updates occur asynchronously in distributed control planes, so it is impossible to compute such a total order.

Instead, we make the observation that a packet traverses routers in the inverse order of route advertisements. Consequently, a packet and a route update currently propagating through the network will "collide" at some router. Any routers the packet traverses *before* this collision will not have seen the route update, and hence the FIBs at these routers will contain the entries that existed prior to the update. Any routers the packet traverses *after* this collision will have seen the route update, and hence the FIBs at these routers will have (if necessary) been updated based on the route update. Thus, to obtain a consistent snapshot—i.e., one that reflects the FIB entries a packet would encounter as it traverses the network at a specific instance in time—we simply need to ensure that *if a FIB snapshot from one router (R) was taken after applying a route update (U), then the FIB snapshot from every other router that had previously received U must also have been taken after applying U.*

We can easily use an HBG to identify which routers must have received a route update, and hence which routers we must receive FIB updates from in order for the snapshot to be consistent. In particular, assume the HBG contains a FIB update for prefix P on router R, and one of the parents of this FIB update is an advertisement for P received by R. We know from our list of common HBRs (§4.1), that a router R' must have sent the advertisement for P before R received it. Consequently, the HBG must contain this output. If the HBG doesn't contain such an output, then all router I/Os have not been received and integrated into the HBG, so we may be missing some FIB updates. If the HBG contains such an output for router R', then we know the FIB updates from R' have been received. If there is a FIB update for P on R' that depends on a route advertisement for P, then we repeat the process to make sure we have seen all I/Os from a router R'' that sent the advertisement to R'. The process repeats until some FIB update for P does not depend on a route advertisement, or the router from which the update was received is external to the network.

Distributed verification. Existing data plane verifiers are centralized: they gather the snapshot of the data plane on a single machine for verification. This approach makes sense for offline analysis and in SDNs with centralized control planes. However, in a distributed control plane, having a centralized verifier reduces robustness and scalability.

Fortunately, several data plane verifiers [22, 23, 24] can be readily transformed intro distributed verifiers. The basic idea is to pass partial verification results between network routers (or a distributed set of verification nodes) and have each router uses its local FIB snapshot to conduct parts of the verification. For example, with HSA [23], each router could maintain its own transfer function and send the output of the transfer function to downstream routers that would apply their transfer functions. This approach adds time overhead, due to the delay in passing partial verification results between routers, but the approach avoids the potential for bottlenecks at a centralized verifier.

Construction and analysis of the HBG can also be distributed. In particular, each router can store its own happens-before subgraph containing that router's control plane I/Os. Partial paths through the HBG can be passed to neighboring routers that can expand the paths based on their happens-before subgraph.

all possible sequences of FIBs a packet could encounter, thereby provide a means to verify per-packet policy compliance.

6 REPAIRING POLICY VIOLATIONS

As discussed in §2, addressing a policy violation by simply blocking problematic FIB updates can lead to other problems. Instead, we want to address the root cause of the violation. In this section, we discuss ways in which HBRs can be used for automatically repairing the control plane, in increasing order of sophistication.

Reverting the root cause event, prior to installing any problematic FIB updates. The HBRs we observe are the sequence of control plane events that led the network to enter a problematic state. By traversing the HBG starting from a problematic FIB update, we can determine the sequence of I/Os that led to the policy violation. Any leaf nodes we encounter represent the root cause(s) of the event. For example, if we traverse the HBG in Fig. 4 starting from the vertex "R_1 install $P \rightarrow Ext$ in FIB", we will reach the leaf node "R_2 configuration change," which is the cause of the policy violation in our example (Fig. 2b). We would therefore automatically revert it and report the configuration change as problematic to the operator. If the change was intended, the operator can simply adapt the policy accordingly (e.g., by stating that R_1 should be used instead of R_2). This illustrates another benefit of our approach: it ensures the high-level policy and network-wide configurations implementing it are always in-sync.

Reverting the root cause event, early on in the computation. A more advanced mitigation technique is blocking the root cause event as soon as possible—*prior* to any violation detection. Doing so requires us to be able to reason about the data plane outcome of any input event—i.e. it requires us to be able to "unfold" the HBRs to detect problems before they arrive. Of course, doing so without a model of the control plane is challenging. Our plan here is to leverage the insight that control plane computations tend to be highly repetitive across prefixes. Many destinations are treated alike by the network control plane and can therefore be grouped into few equivalence classes. Studies have shown that even large networks (100K prefixes) often have less than 15 equivalence classes in total [7]. This repetition enables us to automatically

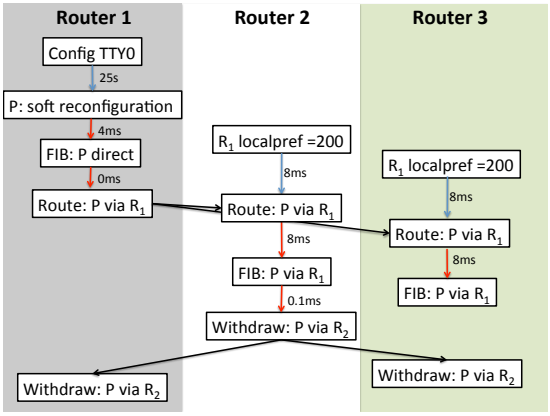


Figure 5: HBG captured from an emulated network using Cisco routers

learn a model of the control plane behavior from the data that we can then use to predict control plane outcomes.

7 FEASIBILITY

To understand the feasibility of our proposal, we deployed an example similar to Fig. 2 in an emulated network [1] containing three Cisco routers (VM images). We enabled logging on all routers and captured and parsed the outputs of the logs. The network starts from a correct state where routers R_1 and R_3 are sending traffic to the external prefix P via router R_2. Next, we manually change the localpref attribute on router R_1 to 200, and manually extract the interesting events from the router logs. The resulting HBG is shown in Fig. 5.

Twenty seconds after the console configuration, router R_1 starts soft reconfiguration: it revisits the routes it has received from its neighbors, re-running the BGP decision process. Very quickly (within 4ms), a direct route to P is installed in the FIB, and after 4ms this route is announced to the other routers. Deriving the HBG on R_1 is fairly straightforward, because the FIB update closely follows the soft reconfiguration. Moreover, the announcement and FIB update are not only close in time, but also target the same prefix, thus suggesting an HBR. The only HBR that is difficult to extract is the one between the TTY config and the software reconfiguration, which are surprisingly far apart (25s) in our emulated network.

On routers R_2 and R_3 the logs begin with the reconfiguration of $localpref$ for R_1, but there are no corresponding logs on R_1 that signal the sending of this information; therefore we have no edges in the HBG. However, the new route received is caused by the route advertisement from R_1; this route gets quickly installed in the FIB (in under 4ms) on both routers, and then R_2 quickly withdraws its own direct route. At this point all three routers have faulty data planes.

The HBG shown in Fig. 5 points to the soft reconfiguration on R_1 as the root cause of the policy violation (and potentially to the TTY0 console changes); this information, coupled with a version system for configurations is enough to allow easy manual rollback, and creates the premises for automated rollback out.

Now consider the timing of updates for the data plane verifier: if it only sees the new FIB from R_3, the verifier will conclude that the path is $R_1 - R_2 - P$, and satisfies the policy; in practice the path will be $R_1 - P$. The verifier misses the problem because it has an inconsistent snapshot of the data plane. Using the HBG, it can catch this inconsistency: the HBG on R_3 contains a route via R_1 that has not been announced in the HBG received from R_1. Consequently, the verifier can wait until it receives the up-to-date HBG from R_1 before verifying the data plane.

8 DISCUSSION

Ensuring networks work correctly as specified by operator policy is an important yet so far elusive goal. In this position paper, we have argued for the need to integrate data plane verification into the control plane in a non-intrusive way. By monitoring control plane input and output we can construct a network-wide happens-before graph (HBG) that helps in reasoning about the provenance of data plane entries and thus enables sound verification and automated repair. We propose to capture FIB updates on all routers and run the verifier to check for correctness before we install updates. Finally, distributed verification is made possible by the data plane consistency information provided by the HBG.

No solution is a panacea, though, and in the following we discuss limitations of our approach. First, we cannot avoid all policy violations: when a route is withdrawn because a link goes down and the withdrawal results in a policy violation, blocking the withdrawal would have no good effects because the traffic would be black holed anyways.

Another limitation is that our approach cannot directly answer *what-if* questions, like control plane verifiers can (with limitations). One approach in this direction is to leverage ideas from CrystalNet [27] that runs an emulated copy of the network and can inject faults.

When repairs are possible, their correctness depends on: *(i)* the precision of the happens-before relationships, to make sure we track down the actual root cause of a violation; and *(ii)* deterministic control-plane execution, to make sure that the control plane will converge to a previously working state given previously seen inputs (i.e., it is memoryless).

Deterministic control-plane execution is particularly important as it enables us to reason about network correctness, without a model, by simply observing that the network was previously correct given a different set of inputs. While routing outcomes are typically deterministic in the case of intra-domain protocols (e.g. OSPF, IS-IS), this is not necessarily true for BGP. Fortunately, BGP determinism can be guaranteed with the help of extra mechanisms such as BGP Add-Path [2], which is widely-available in today's platforms and does not hamper BGP flexibility.

9 ACKNOWLEDGMENTS

This work is supported in part by the Superfluidity H2020 project (Horizon 2020, European Commission) grant 671566 and National Science Foundation grant CCF-1637427.

REFERENCES

[1] GNS3: The software that empowers network professionals.

[2] IP routing: BGP configuration guide, Cisco IOS XE release 3S - BGP additional paths. http://cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xe-3s/irg-xe-3s-book/irg-additional-paths.html.

[3] Quagga routing suite. http://www.nongnu.org/quagga.

[4] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. A general approach to network configuration verification. In *SIGCOMM*, 2017.

[5] 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 *SIGCOMM*, 2016.

[6] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[7] T. Benson, A. Akella, and D. A. Maltz. Mining policies from enterprise network configuration. In *Internet Measurement Conference (IMC)*, 2009.

[8] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo. The good, the bad, and the differences: Better network diagnostics with differential provenance. In *SIGCOMM*, 2016.

[9] Cisco Systems. BGP best path selection algorithm. http://cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html.

[10] Cisco Systems. Cisco IOS debug command reference, release 12.2. http://www.cisco.com/c/en/us/td/docs/ios/12_2/debug/command/reference/122debug/dbfintro.html.

[11] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol version 1.2. RFC 5246, RFC Editor, August 2008. https://www.ietf.org/rfc/rfc5246.txt.

[12] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The matter of Heartbleed. In *Internet Measurement Conference (IMC)*, 2014.

[13] A. El-Hassany, J. Miserez, P. Bielik, L. Vanbever, and M. Vechev. SDNRacer: Concurrency analysis for software-defined networks. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2016.

[14] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev. Network-wide configuration synthesis. In *International Conference on Computer Aided Verification (CAV)*, 2017.

[15] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. D. Millstein, V. Sekar, and G. Varghese. Efficient network reachability analysis using a succinct control plane representation. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[16] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. A general approach to network configuration analysis. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[17] A. Gember-Jacobson, A. Akella, R. Mahajan, and H. Liu. Automatically repairing network control planes using an abstract representation. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[18] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan. Fast control plane analysis using an abstract representation. In *SIGCOMM*, 2016.

[19] A. Gember-Jacobson, W. Wu, X. Li, A. Akella, and R. Mahajan. Management plane analytics. In *Internet Measurement Conference (IMC)*, 2015.

[20] Juniper Networks. Example: Configuring BGP trace operations. https://www.juniper.net/documentation/en_US/junos/topics/topic-map/bgp-troubleshooting.html.

[21] Juniper Networks. Understanding BGP path selection. http://juniper.net/documentation/en_US/junos12.1/topics/reference/general/routing-ptotocols-address-representation.html.

[22] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[23] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[24] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying network-wide invariants in real time. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[25] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. Technical Report 339, 2013.

[26] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[27] 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 *Proc. of the 26th Symposium on Operating Systems Principles (SOSP)*.

[28] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: language, execution and optimization. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2006.

[29] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 289–300. ACM, 2005.

[30] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[31] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King. Debugging the data plane with Anteater. In *SIGCOMM*, 2011.

[32] J. Miserez, P. Bielik, A. El-Hassany, L. Vanbever, and M. Vechev. SDNRacer: Detecting concurrency violations in software-defined networks. In *ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*, 2015.

[33] L. Ryzhyk, N. Bjørner, M. Canini, J.-B. Jeannin, C. Schlesinger, D. B. Terry, and G. Varghese. Correct by construction networks using stepwise refinement. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[34] R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu. SymNet: scalable symbolic execution for modern networks. In *SIGCOMM*, 2016.

[35] K. Subramanian, L. D'Antoni, and A. Akella. Genesis: Synthesizing forwarding tables in multi-tenant networks. In *SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2017.

[36] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock. Scalable verification of border gateway protocol configurations with an SMT solver. In *ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, (OOPSLA)*, 2016.

[37] G. C. Whittaker. Network outages like NYSE, United Airlines, are the new natural disasters. http://bit.ly/1HW9wgr, July 2015.

[38] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems with negative provenance. In *SIGCOMM*, 2014.

[39] W. Zhou, D. K. Jin, J. Croft, M. Caesar, and P. B. Godfrey. Enforcing customizable consistency properties in software-defined networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[40] W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr. Distributed time-aware provenance. In *International Conference on Very Large Data Bases (VLDB)*, 2013.

[41] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2010.