Accuracy, Scalability, Coverage – A Practical Configuration Verifier on a Global WAN

Fangdan Ye[★△‡], Da Yu[§△‡], Ennan Zhai[†], Hongqiang Harry Liu[†], Bingchuan Tian[×△], Qiaobo Ye[†]
Chunsheng Wang[†], Xin Wu[†], Tianchen Guo[†], Cheng Jin[†], Duncheng She[†], Qing Ma[†]
Biao Cheng[†], Hui Xu[†], Ming Zhang[†□], Zhiliang Wang[★□], Rodrigo Fonseca[§]
[†]*Alibaba Group* [★]*Tsinghua University* [§]*Brown University* [×]*Nanjing University*

ABSTRACT

This paper presents Hoyan– the first reported large scale deployment of configuration verification in a global-scale wide area network (WAN). Hoyan has been running in production for more than two years and is currently used for all critical configuration auditing and updates on the WAN. We highlight our innovative designs and real-life experience to make Hoyan accurate and scalable in practice. For accuracy under the inconsistencies of devices' vendor-specific behaviors (VSBs), Hoyan continuously discovers the flaws in device behavior models, thus aiding the operators in fixing the models. For scalability to verify our global WAN, Hoyan introduces a "global-simulation & local formal-modeling" strategy to model uncertainties in small scales and perform aggressive pruning of possibilities during the protocol simulations. Hoyan achieves near-100% verification accuracy after it detected and fixed $O(10)$ VSBs on our WAN. Hoyan has prevented many potential service failures resulting from misconfiguration and reduced the failure rate of updates of our WAN by more than half in 2019.

CCS CONCEPTS

• **Networks** → **Network reliability**; • **Theory of computation** → **Logic and verification**;

KEYWORDS

Network Verification; Network Configurations; Reliability

ACM Reference Format:
Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, Duncheng She, Qing Ma, Biao Cheng, Hui Xu, Ming Zhang, Zhiliang Wang, Rodrigo Fonseca. 2020. Accuracy, Scalability, Coverage – A Practical Configuration Verifier on a Global WAN. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3387514.3406217

[‡]Both authors contributed equally to the paper.
[△]Work done while these authors were interns at Alibaba Group.
[□]Co-corresponding authors.

SIGCOMM '20

ACM ISBN 978-1-4503-7955-7/20/08…$15.00
https://doi.org/10.1145/3387514.3406217

1 INTRODUCTION

Alibaba has a global network infrastructure to serve over one billion customers world-widely with its online services, such as AliCloud (cloud service), Taobao/Tmall/AliExpress (e-commerce), Alipay (FinTech), and so forth. It builds and operates a global wide area network (WAN) to inter-connect its tens of data centers in four continents and peer with external Internet Service Providers (ISPs). This WAN contains hundreds of routers and realizes connectivity demands of applications via complex static routes, border gateway protocol (BGP), and intermediate system to intermediate system (IS-IS) configurations as well as access control list (ACL) rules.

Maintaining the reliability of such a large-scale and complicated WAN is extremely challenging, and one major source of risk is the obscure errors in routing configurations that violate operators' reachability intent. To prevent incidents resulting from routing configuration errors, we decided to adopt network verification techniques as an automatic and systematic way to proactively audit our configurations and prevent configuration-induced incidents. Nonetheless, despite that there were multiple existing solutions [4, 11–13] when we started to build our WAN configuration verifier, Hoyan, in late 2017, we found it was far from straightforward to directly apply them into our WAN environment due to both pragmatic challenges of usability and the fundamental trade-offs faced by existing network verifiers.

Pragmatic challenges. The primary challenge is the *accuracy* of control plane modeling. An essential precondition of the usefulness of a configuration verifier is itself is correctly implemented, and operators have confidence in its results. However, this issue is less discussed by the previous literature. In practice, besides the potential bugs in software developments, the most crucial challenge of accuracy is that the correctness of the behavior model of network devices is hard to guarantee due to the existence of vendor-specific behaviors (VSBs). Specifically, different vendors implement parts of a protocol in various ways which might not be known for developers when building the verification tools. For example, in BGP, `remove-private-AS` in Vendor A's implementation means "removing all the private AS numbers", but means "removing only private AS numbers until the first non-private one" in Vendor B.[1] Since routing behaviors of devices are highly correlated, failing to catch one or several VSBs can significantly invalidate the verification results. For instance, before taking into account VSBs, the verification accuracy of some prefixes in our network can be less than 20% (Figure 14). Therefore, without a systematic methodology that continuously enhances the accuracy of our configuration verifier, the verification result can hardly be trusted by operators.

[1]We use examples from real vendors, but omit the vendor names.

Fundamental trade-offs. Prior work has a fundamental trade-off between scalability and verification coverage (*e.g.*, handling of failures, handling of route update racing, supported protocols, *etc.*). For example, simulation-based verification systems (*e.g.*, Batfish [12]) simulate the entire process of route propagation and convergence on the control plane and check the forwarding tables (FIBs) over a given network snapshot. Because these systems merely verify one data plane under one topology each time, they have to run $\binom{n}{k}$ times to verify configurations under arbitrary k failures out of n links [4], even though they archive good scalability in the $k = 0$ cases. Similarly, the simulation-based verification cannot check configurations under different arrival orders of routes with good scalability. Alternatively, logical-formula-based approaches (*e.g.*, Minesweeper [4]) offer the capability of reasoning about arbitrary k failures and all possible arrival orders of route updates. They are, however, unscalable since they represent the configuration logic of the entire (or a large part of the) network as a single, big logical formula with diverse constraints – solving such a formula (even with a modern SMT solver) is impractical [5, 24]. In addition, while recent efforts improve the scalability of existing tools by taking advantage of network topology symmetry [5, 23, 24], such a benefit is limited on our WAN, as our WAN lacks this symmetry. While graph-based tools, *e.g.*, ARC [13], are fast, they are not able to encode complicated routing policies [4]. Therefore, scalability and verification coverage (*a.k.a.* more advanced features) are hard to co-exist in existing tools. (See more discussions in §2).

In this paper, we highlight several innovative designs and real-life experiences that make our verification system, Hoyan, accurate and scalable with sufficient coverage on uncertainties and protocols.

Accuracy. Hoyan continuously compares the routes it computes with the actual ones in production, testbeds or emulations to detect the flaws of its behavior models. One major challenge is how to locate the root cause of a reachability mismatch to facilitate model tuning. With existing network monitoring tools, it is possible to locate the root cause of a wrong place because a VSB's impact typically appears far away from the actual location. To discover differences between its model and the real device, Hoyan combines all the attributes of a route relevant for routing into an extended routing information base (RIB). Hoyan is able to locate the first place the mismatch happens by comparing each of the attributes. As a result, Hoyan can accurately locate a VSB within $O(10)$ configuration lines. After that, developers of Hoyan can easily find the corresponding configuration block and produce patches to improve the verification accuracy. Another challenge is the difficulty to validate behavior models of Hoyan against the actual device behaviors under arbitrary cases. Instead, our strategy is to validate behavior models under all cases that appear in the production. For a given device type, we find all places in production where the device type sits and validate Hoyan 's model with production data.

Scalability with sufficient coverage. Hoyan offers a *globally simulation-based & locally formal-modeling-based* solution to *simultaneously* take advantage of the scalability of simulation-based solutions and the ability to handle uncertainties of formal-modeling-based solutions. Specifically, Hoyan runs overall simulations of all protocols on the WAN which are fast and lightweight compared with formal-modeling-based approaches. On the other hand, Hoyan introduces small-scale logical formulas to encode all possibilities when it encounters uncertainties within considerations and solves small-scale SMT problems during or at the end of the simulations to verify reachability under uncertainties as well as preserving scalability. For example, when simulating the propagation of route updates, Hoyan incrementally encodes, with a logical formula, the topologies under which a route update can reach a device, or under which a rule exists in the routing table. Hoyan leverages this encoding for its substantial scalability improvements. For instance, Hoyan traces the process of route update or packet propagation, so that it can cut unnecessary propagation branches whose topology condition is impossible or out of consideration (larger than k failures). This pruning cannot be done by current formal-modeling-based tools since the latter do not have access to the intermediate states during route propagation.

Real-world deployment. Hoyan has been deployed on our WAN for more than two years and prevented many incidents resulting from misconfiguration. The overall rate of update-triggered network incidents was cut by more than half in the second year of its deployment. In §7, we share some representative cases of configuration errors and the VSBs we found in practice and our experience with other practical challenges.

Performance. In §8, we compare the performance of Hoyan with representative alternatives (Batfish [12], Minesweeper [4], and Plankton [24]), with realistic network setups. Hoyan outperforms them by orders of magnitude in large scale production WAN.

2 RELATED WORK

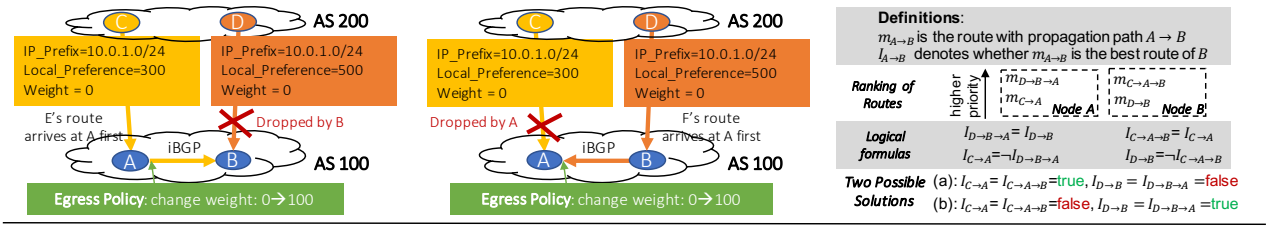
The core idea of network verification is leveraging formal methods to validate the network behavior. Depending on how to model network behaviors, there are three strategies.

Data plane verification. Data plane verification systems focus on modeling FIB snapshots to logical formulas and then checking reachability properties via solvers. There have been many data plane verification systems, *e.g.*, NoD [21], Anteater [7], Veriflow [19], HSA [18], Delta-net [14], SymNet [27], and P-Rex [16]. Microsoft's RCDC [15], to the best of our knowledge, is the first publicly-reported data plane verification system deployed in the data center scale. We do not take this path because our goal is to *proactively* detect errors before adopting potentially buggy configurations in the network. Also, data plane verification does not provide solutions to uncertainties, *e.g.*, failures and non-deterministic errors.

Configuration verification. Configuration verification (or control plane verification) systems aim to *comprehensively* verify the logics encoded by the configurations of diverse routing protocols. Existing efforts can be classified into three categories:

(i) *Simulation-based verification:* Batfish [12], C-BGP [25], and FastPlane [22] take as input network configuration and multiple given environments, simulate the control plane, and finally generate the corresponding data planes. They, then, leverage the existing data plane work (mentioned above) to check misconfiguration. Simulation-based efforts suffer from scalability issues to take into account uncertainties (*e.g.*, k-failure tolerance) in large networks, because they have to enumerate all possible cases.

(ii) *Formula-based verification:* ERA [11] and Minesweeper [4] build the network model based on the entire configuration and



(a) Original Intent: Adding a rule to enlarge the weight of the route update from A to B, so B will select A as the next-hop rather than D, and A and B will forward packets of 10.0.1.0/24 to C (larger weight overrides the larger local preference).

(b) Error from route update racing: the route from D arrives A before C's, A selects D's route first and directly drops C's, so that C's route gets no chance to hit the weight changing rule. So, A and B still forward packets of 10.0.1.0/24 to D.

(c) Meaning of the logical formulas: a route is selected if its predecessor is selected and the routes ranked higher are not selected. **Multiple solutions of the formulas imply ambiguous route convergences**. Solution (a) represents the case in (a), and Solution (b) represents the case in (b).

Figure 1: A simplified real example of configuration bug caused by non-deterministic route update racing.

then convert network properties verification into SMT and BDD problems, respectively. These solutions suffer from scalability problems in the large network, because they encode the entire control plane logic into a single SMT/BBD formula. This formula can easily be overly large for any SMT/BBD solver to compute within a reasonable duration. Recent efforts improve the scalability of formal-modeling tools by taking advantage of network topology symmetry [5, 23]. While these proposals might work well in highly symmetric networks such as data center networks (DCNs), their benefits are limited on our WAN, as our WAN lacks this symmetry, as other WANs might also do. Plankton [24] was recently proposed to get better scalability with model checkers and some optimizations to reduce the problem size. It supports route update racing, but it is not scalable to handle failures without topology symmetry. In addition, Bagpipe [29] shares similar ideas, but only focuses on BGP configuration verification. Microsoft's SecGuru [15] and Alibaba's Jinjing [28] focus on ACL rules rather than routing configurations.

(iii) *Graph-based verification:* Efforts, *e.g.*, ARC [13] and Tiramisu [1], model configurations as a directed graph, and then use graph-based algorithms to check the network properties. ARC is limited to encode complicated BGP routing policies. Tiramisu [1] improves ARC to handle communities in BGP with multi-layered graphs, but it cannot support route selections with policies or route update racing.

Hoyan is the first reported large scale deployment of configuration verification in production. It has two innovations that make it pragmatic in the wild. First, it systematically handles VSBs to make the verification result to be trustworthy, which is a foundation; Second, it creates a "globally simulation & locally formal-modeling" strategy to achieve good scalability with the ability to cover some critical uncertainties, which is useful in practice (see details in §3.2).

Network emulation. Network emulators, *e.g.*, CrystalNet [20], run the real control plane software in emulated environments and generate corresponding FIBs for validation purposes. It is essentially an enhanced version of simulation-based verification which can handle VSBs and other software bugs. Compared with verification, network emulation has *two* issues which make the configuration verification irreplaceable. First, the network emulation needs vendors to provide their device firmware within virtual machines or containers, while we found it is practically hard to get such support from all vendors for all device models in Alibaba's WAN at present. Second, running real router software requires a large number of computing resources (*e.g.*, $100 per hour for emulating just one data center [20]). As routers in WAN typically have much more sophisticated firmware than data center switches, running emulations for a WAN can be even more expensive. Therefore, emulation is useful but not available all the time, and verification thus is still an effective and lightweight tool to validate the logical correctness of network configuration.

3 OVERVIEW

This section first introduces Alibaba's global WAN (§3.1), and then shows our motivations (§3.2) and goals (§3.3).

3.1 Alibaba's global network infrastructure

Alibaba has a global scale infrastructure to support its various types of online services (*e.g.*, computation, storage, search, video, *etc.*), which have more than one billion users in total. By January 2020, this infrastructure has $O(10)$ data centers, $O(100)$ PoP (point of presence) nodes, $O(1000)$ edge sites across 20 geographical regions over North America, Europe, Asia, and Oceania.

Alibaba operates a private WAN that interconnects all its data center networks (DCNs) and peers with external ISPs around the world. The DCNs and the WAN employ eBGP for interconnecting; the WAN is a single AS that uses iBGP on top of IS-IS to internally redistribute routes learned from outside. The whole WAN has $O(100)$ routers, which are from multiple vendors, and $O(1000)$ links. Each router has $O(1000)$ lines of configuration commands. There are also $O(10,000)$ IP prefixes announced over the WAN. With the fast growth of the application demands, our WAN doubles its scale every year, and performs $O(1000)$ update operations per month.

3.2 Our motivations

Two inherent characteristics of our WANs make it arduous to manage network configurations. First, our WAN is incrementally built for years, so its current configuration is an accumulated set of historical configuration updates from different operators in different scenarios as the WAN continuously grew into the current one. Therefore, one big concern of us (and what we have seen in practice) is the current configuration on the WAN might have some hidden inconsistencies or bugs which can be triggered out of the blue. Reasoning about the risks in existing configurations – especially finding unexpected reachability violations under hardware (*e.g.*, device and link) failures (called k-failure tolerance in Beckett *et al.* [4]) – is extremely hard even for well-trained operators.

Second, we update the configurations of our WAN on a daily basis because of either the applications' footprint expansions or

Requirement	Property of verification tool	Batfish [12] (Simulation)	Minesweeper [4] (Formal-Modeling)	ARC [13] (Graph Modeling)	Hoyan (Global Simulation & Local Formal-Modeling)
Mandatory	Scalability of computations	✓	✗	✓	✓
	Correctness with vendor's heterogeneity	✗	✗	✗	✓
	Comprehensiveness of protocols	✓	✓	✗	✓
Preferred	Handling failures of router/link	✗	✓	✓	✓
	Handling route update racing	✗	✓	✗	✓
Optional	General route inputs	✗	✓	✗	✗

Table 1: The properties of verification and representative solutions when Hoyan was designed in late 2017.

the optimizations of the WAN's architecture or types of equipment. Performing the updates on configurations correctly and quickly is critical to our business, but it is also unsurprisingly hard and error-prone. According to our operation experiences, severe incidents can be triggered by even a tiny flaw in the new configuration, including but not limited to incorrect policy designs, confusions of new or abandoned features, VSBs, typos, and so forth. In addition, the new configuration may have ambiguities due to the non-deterministic orders of route update arrivals in update periods (shown in Figure 1), which is extremely hard to trace manually.

Hence, as our WAN gets larger and more complex, and our applications change their requirements on networking more frequently, we are strongly motivated to adopt verification as an automatic, rapid and comprehensive way to (i) audit the current configuration snapshot for hidden errors; and (ii) check correctness and inconspicuous ambiguities of new configurations in an update.

3.3 Our goals

Building a pragmatic and effective configuration verification tool on a large-scale and complex WAN like ours requires systematic reasoning about the trade-offs we face and what need be achieved and what need not according to practical concerns. Based on the experience of our operators, Table 1 lists six key properties for configuration verification, and classifies them into three groups.

Mandatory. The properties in this group are the most essential ones to make the whole system applicable to the WAN. For scalability, our goal of the waiting time to get a verification result is no more than 24 hours on a standard commodity server. Simulation-based solutions achieve good scalability, since simulating control plane protocols is lightweight and easy to be parallelized. For instance, Batfish can finish the verification of a configuration snapshot in about 1 hour. But Minesweeper, as a classic formal-modeling based solution, cannot finish within 24 hours even on a subnet smaller than 10% in size of our WAN (§8.2), because its SMT formulations become too large when the network has a large number of peering sessions, per-interface policies, and IP prefixes.

The second mandatory property is the correctness of verification results, especially under the heterogeneity of different vendors. In practice, different vendors usually implement routing protocols with their own understandings and definitions, which is hard to know ahead of time. Such VSBs can trigger cascading errors in the verification computations, resulting in verification results that are far from trustworthy. For instance, before we consider the effects of VSBs, our verification results have poor accuracy: 79% IP prefixes have less than 60% accuracy rate compared with the ground-truth (§6). Hence, a practical configuration verifier must effectively detect VSBs and model the behaviors of devices in a vendor-specific way. Unfortunately, no existing verification tool, we are aware of, clearly demonstrates how to deal with the issues with VSBs.

The third property is protocol coverage. We aim to support eBGP, iBGP, IS-IS, static route, and the route redistributions among these protocols correctly so that efforts like ARC fall off the map.

Preferred. The properties in this group are features that are very helpful to enhance the power of configuration verification, though they are not fundamentally indispensable for a simple purpose of verifying a static configuration snapshot.

The handling of uncertainty due to router and link failures is one such a property. This property can enable operators to understand the reachability of the network under some degree of failures (*e.g.*, up to k link failures) and help to design more robust configurations.

Besides hardware failures, the non-deterministic racing of route updates is another source of uncertainty, which operators are interested to understand. Specifically, operators find that some incorrect configurations can result in different converged routes over the network under different orders of route update arrivals. For instance, Figure 1 (a) and (b) illustrate an iBGP configuration error triggered by the race of route updates, which is a simplified version from a real case. Therefore, if the configuration verifier can find the risks caused by races of route updates, operators can control the routing across the network more precisely and safely.

Optional. This group means the included properties are not necessary to our network operations. They can be given up to make the solution more lightweight or make other properties easier to realize. General route inputs, as shown in Table 1, mean whether the verification methodology is able to check properties of interest under arbitrary route conditions, such as "in what route inputs the BGP route hijacking would occur" and "in what route inputs, A and B are isolated". To handle the general route inputs, the simulation-based and graph-based verification systems (*e.g.*, Batfish and ARC) need to emulate all the cases which is very time-consuming. Minesweeper is efficient in solving the general route inputs case since it can encode the general route inputs into symbolic formulas.

4 HOYAN ARCHITECTURE

Hoyan offers verification services for network operators to check whether a set of configurations meets some particular properties; meanwhile, it constantly compares the routes it computes from its current device behavior models (detailed in §4.2) and online configurations against the routes in real networks, to find flaws in its current behavior model, facilitating the operators to fix these flaws. Hoyan thus has two parts, as shown in Figure 2.

In the frontend, the *configuration verifier* (§5), combines the current online configuration and the proposed configuration changes from operators to generate the target configuration to be verified. According to the stock keeping unit (or SKU) of each device of the network, it obtains each device's behavior model and feeds them with the target configuration to generate the target network

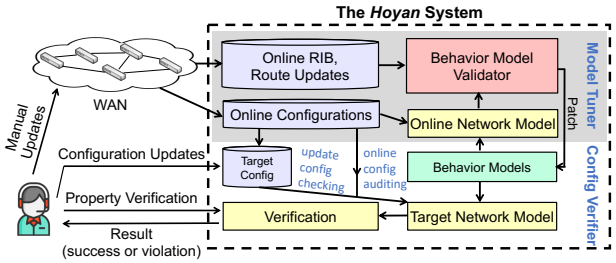


Figure 2: Hoyan's architecture.

model. After that, the verification block queries the target network model to answer the operator's verification questions. Hoyan offers user-friendly interfaces for our operators to express the property of interest. Based on our operation needs and experience, Hoyan mainly focuses on verifying the properties of reachability and role equality. Our WAN has no requirement for verifying properties such as isolation issues or black holes which mainly exist in DCNs.

In the backend, the *behavior model tuner* (components in the gray box) continuously collects online configurations and network information, including routing table (RIB) and route updates. Alibaba has internal systems that record and maintain the above information. The tuner, on one hand, passes these configurations to each device's behavior model to generate the online network model (for the entire WAN). On the other hand, it calls the validator to check whether there are any mismatches between the model it computes and the network information it collects. When a mismatch is found, the tuner localizes to a small configuration snippet, so that a human can easily understand. Network operators normally write a small patch to fix the flaw in corresponding device models (§6).

In the rest of this section, we first explain the key ideas behind Hoyan design (§4.1), and then describe how Hoyan models device and network (§4.2).

4.1 Key ideas behind Hoyan design

We choose to start from the strategy of simulation like Batfish, because it can satisfy our needs in "Mandatory" if it is patched with VSBs handling. Surprisingly, we also find that "Simulation" can also be enhanced with "Formal-Modeling" to handle the uncertainties (*i.e.*, the k-failure tolerance and non-deterministic route updating racing). We therefore call the design strategy of Hoyan as *global simulation & local formal modeling*, and it can achieve all properties in both "Mandatory" and "Preferred" as shown in Table 1.

Handling VSBs and uncertainties in simulation-based configuration verifiers while keeping their scalability and comprehensiveness are the two major innovations in Hoyan.

Handling VSBs. To perform its simulations, Hoyan needs models of the behavior of routers. The key idea to make sure behavior models align with the real implementation of vendors is to continuously compare the RIB/FIB Hoyan gets from simulations and the ground truth inside production networks and repair the behavior models to approach the ground truth. To realize this idea, we must (i) make sure all device types can be tested with sufficient cases; (ii) provide enough hints for Hoyan to quickly locate the deficits of the behavior models and repair them (detailed in §6).

Handling uncertainties. The reason formal-modeling-based approaches can handle uncertainties is that logical formulas can easily

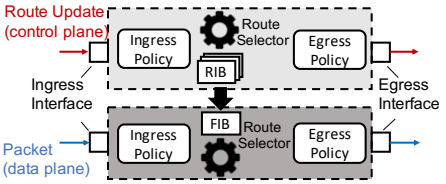


Figure 3: A device behavior model.

encode "if-else" logics. For example, Minesweeper uses a binary symbolic variable that indicates the liveness of a link and takes it into the formula. The formula specifies what to do if the link is up or down. Similarly, our key idea to handle uncertainties in simulations is to use logical formulas to encode uncertainties in some small steps during the simulations. Given the small scale of encoded model in these steps ($O(100)$ variables on average per formula), an SMT solver – Hoyan employs Z3 [10] as its SMT solver – can quickly get the results. This strategy thus is called *Global simulation & Local formal-modeling*.

For link and router failures, when simulating the propagation of route updates or packets, Hoyan constructs a logical formula to *encode* the topologies under which a route update or packet can reach a device, or under which a rule exists in the RIB/FIB. With this encoding, Hoyan traces the process of route update or packet propagation, so that it can cut unnecessary propagation branches whose topology condition is impossible or out of consideration, *e.g.*, larger than k failures. This pruning can significantly accelerate the verification process (detailed in §5).

For non-deterministic route update racing, Hoyan uses binary symbolic variables to indicate the orders of route update arrivals on each router, so that it can encode the selected routes under arbitrary arrival orders of route updates within a single simulation process. At the end of the process, it just needs to solve an SMT formulation to check whether there exist multiple arrival orders. If so, that means non-deterministic route update racing exists.

4.2 Modeling devices and network

This section explains two important concepts: device behavior model and network model.

Device behavior model. As shown in Figure 3, a device behavior model consists of two pipelines for processing route updates on the control plane and packets on data plane respectively (we use the word "message" to refer to a route update or a packet). A concrete device behavior model is generated from the device configuration and the vendor specific behavior modeler of the device type. Hoyan builds vendor specific configuration parsers and behavior modelers for all types of devices that could appear in production networks.

Each processing pipeline has three sequential components: ingress policy, route selector, and egress policy, as shown in Figure 3. Ingress and egress policies are essentially match-action tables that define whether to forward or drop a message and/or how to modify a message based on the pattern of the message. For instance, on the control plane, a BGP router can have an ingress policy which drops route updates from a particular peer, and on the data plane, a router can have an egress ACL rule that drops UDP packets on a particular interface. The route selector encodes the core logic of routing protocols on the control plane or the forwarding logic on the data plane. For instance, in a BGP router, the route selector decides how

to prioritize routes from different peers for the same subnet and sends the best route to the peers of this router; on the data plane, it matches a packet to a rule in the FIB based on longest prefix or other built-in logic and selects next hop(s) to forward the packet.

For different vendors and different device SKUs, the specific behavior of ingress and egress policies and the logic inside route selectors can be distinct. Therefore, Hoyan must build device behavior models adaptively.

Network model. After obtaining all the behavior models of network devices, Hoyan connects them according to the network topology. If two devices have a link between them, their ingress and egress modules in their behavior models are connected together in both directions. The resulting graph is called *network model*.

In a network model, Hoyan first makes route announcements from the edges of the network model. Each route update is processed by the control plane pipeline it meets, so that it is propagated across the network model. The RIB of each device is established once the propagation process is done. The FIB then is derived from the RIB. With the complete RIB and FIB, the reachability of a given message can be easily evaluated by combining relevant logical rules along the pipelines the message encounters.

5 VERIFICATION METHODOLOGY

This section shows how Hoyan performs reachability verifications and achieves the tremendous improvement in scalability.

5.1 Reachability on control- and data-planes

According to the requirements from operators, Hoyan checks the reachability of both routes and packets. On one hand, Hoyan justifies whether the route to a given subnet can reach a group of network devices after the route propagation process. This is also useful for debugging configurations such as BGP peering, routing policies, *etc.* On the other hand, Hoyan can also check whether packets towards a given subnet can start from a group of network devices and reach the gateway router of the subnet. Note that "the route of a subnet can reach device A" does not necessarily mean "a packet can reach the subnet's gateway router from A" because of multiple reasons such as data plane ACL rules and longest prefix matching.

5.2 Intuitive example of topology condition

We first use an intuitive example to explain the concept of *topology condition encoding*, which is the key to Hoyan's good scalability in verification with failure cases.

Whether a device sends or receives a message depends on the up or down status of particular links. Figure 4 shows a simple BGP network. We use a tuple (Subnet, AS Path, Nexthop, TopoCond) to denote a BGP update with corresponding topology condition. A topology condition is a logic formula composed of binary link *aliveness* variables (a_n). For example, a_1 = *True* means Link1 is up. At step ①, C receives a BGP update that originated from A, (N, 100, A, a_1), if Link1 is alive. At this step, a_1 is the *topology condition* which must be true for C to get the route update from A. Similarly, at steps ② and ③, the route updates m_2 and m_3 also have their topology conditions: m_3's topology condition is $a_2 \wedge a_3$, as B must first receive the route under a_2 *and* then forward to C under a_3.

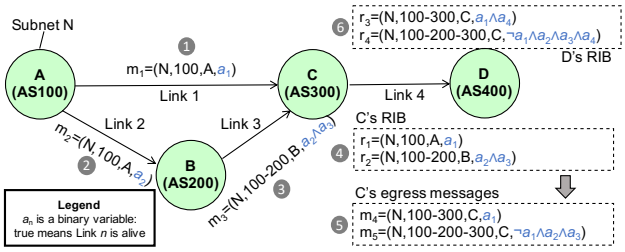


Figure 4: Route update & RIB with topology conditions.

After receiving messages (m_1 and m_3), C writes them into its routing table (RIB) (step ④). Rules in the RIB are directly inherited from the topology conditions in the messages. C's RIB has two rules (r_1 and r_2), whose topology conditions are inherited from m_1 and m_3 respectively. Following BGP, C ranks all received routes and only forwards the best to its peer D (and B). In this case, C ranks r_1 higher because it has a shorter AS Path. If r_1 exists, C sends m_4 to D. Otherwise, it sends m_5 (step ⑤). Note m_5's topology condition is the negation of r_1's in conjunction with r_2's. This means under the case that r_1 does not exist in C's RIB but r_2 does. Finally, D receives m_4 and m_5 and builds its RIB (r_3 and r_4) with the corresponding conditions (step ⑥).

After this route propagation process, it is straightforward to check the route reachability from A to D. For example, the topology condition for D to receive at least one route for Subnet N is r_3's topology condition or r_4's topology condition. It is also easy to find the failure case with the least link failures which causes unreachability from A to D. In this case, failure of Link 4 ($\neg a_4$) makes D unreachable from A.

5.3 Topology condition encoding

Topology condition encoding in messages. Formally, given a route update m, we use a logical formula ($I(m, A)$) to decode the topology condition that message (m) reaches the ingress pipeline of device A. Similarly, we use $E(m, A)$ to refer to the topology condition that message (m) is sent to the egress pipeline of device A. $I(m, A)$ and $E(m, A)$ are composed of the link aliveness variables.

Topology condition encoding in RIB/FIB. In real networks, a device will select routes from what it receives to build up its RIB. In Figure 4 example, we have two observations: (i) the ranking of the routes only depends on the properties of the routes, independent with the topology conditions; and (ii) a lower ranked route will be selected if the ones in higher ranks are missing in the device. Thus, we extend the real world RIB to have all received routes of a device with ranking and their corresponding topology condition.

In Figure 4, an implicit condition for inserting m_1 and m_3 into C's RIB is that they pass C's ingress policy. The ingress policy only considers the attributes of the route updates and is independent of topology conditions. Thus, r_1 and r_2 can keep m_1 and m_3's topology conditions respectively if they survive after C's ingress filtering.

In general cases, a rule in RIB is directly derived from a route update. However, in practice, route aggregation, which merges several small subnet routes into a single larger subnet route, is common. To handle route aggregation, Hoyan reads the trigger conditions of route aggregation from configurations and separately handle the cases with and without route aggregation. For instance, if the configuration indicates that routes for "10.0.1.0/32" (with ingress

topology condition I_1) and "10.0.1.1/32" (with ingress topology condition I_2) will be aggregated to a single route for "10.0.1.0/31" once both of them are received. We put the following rules in the RIB:

$$r_{agg} = (10.0.1.0/31, *, *, I_1 \wedge I_2)$$
$$r_{sub1} = (10.0.1.0/32, *, *, I_1 \wedge \neg I_2)$$
$$r_{sub2} = (10.0.1.1/32, *, *, \neg I_1 \wedge I_2)$$

Note that r_{agg}, r_{sub1}, and r_{sub2} are exclusive with each other in topology conditions. Despite in theory this method to handle route aggregation can lead an exponential number of case combinations, we find it works well in practice because operators usually explicitly write a moderate number of aggregation triggering cases in the configuration and forbid automatic route aggregations for the controllability to routes.

5.4 The reachability of routes

The reachability of a route r to a network device D means that whether D can receive a route r ultimately when the control plane protocols have converged. The operators can specify a particular route, *e.g.*, ("10.0.1.0/31", 100-200-300, C), or a pattern representing a group of routes, *e.g.*, ("10.0.1.0/31", *, *) which means any route to subnet "10.0.1.0/31" no matter the AS path or the next-hop, to verify the reachability.

The key design of Hoyan on (1) checking reachability under failures and (2) detecting non-deterministic route update racing is to derive topology conditions for each route update and rule in RIB.

Iteratively deriving topology conditions of routes. We use three simple rules to iteratively derive the topology conditions of all route updates, rules in RIB:
(i) *From RIB to egress*: Suppose r^i denotes a rule in RIB and $r^1, \ldots, r^{i-1}$ are rules to the same destination subnet with higher priority than r^i. The topology condition to send a route update (m^i) from r^i to egress policy pipeline is $\neg R(r^1) \wedge \ldots \wedge \neg R(r^{i-1}) \wedge R(r^i)$, in which $R(r)$ is the topology condition of rule r in the RIB. For example, in Figure 4, the topology condition of m_5 is computed according to this rule. Note that one implicit condition for the transition from a RIB rule to a route update is that the route selector decides to send a route to the egress interface.
(ii) *From egress to other side's ingress*: For a route update m with topology condition $E(m, S)$ in S's egress, the topology condition for m to reach D's ingress (D is a peer of S) is $I(m, D) = E(m, S) \wedge a_l$ where l is the link from S to D. For example, in Figure 4, the topology condition of m_3 is derived from m_2 based on this rule. Note that one implicit condition for the transition from egress to the other side's ingress is that m passes the egress policy of S.
(iii) *From ingress to RIB*: For a route update m with topology condition $I(m, D)$ in D's ingress, the route selector applies a route sorting algorithm to put m into RIB as rule r^i with topology condition $I(m, D)$ if no route aggregation is triggered. In Figure 4 example, r_1 and r_2's topology conditions are from m_1 and m_3. Otherwise, route aggregation will be applied and new rules will be available in RIB hereafter as described earlier. Again, one implicit condition for the transition from ingress to RIB is that m passes D's ingress policy.

With the preceding three simple rules, starting from the configured routes (with True topology condition) in the RIBs of the gateway routers of all subnets, Hoyan can iteratively derive the route updates and the RIBs with their corresponding topology conditions. The whole route propagation process ends when no router has any new route updates to send.

During the route propagation process, one critical issue is to handle "late higher priority routes". Specifically, the topology condition of a route update generated from a low priority RIB rule depends on high priority rules. Therefore, if a lower prioritized rule arrives at a device first, it is possible that it is announced with an incorrect topology condition. The solution to this issue is to track the propagation of each route and make an adjustment to the topology condition of the lower priority route by "anding" the negation of the new, higher prioritized rule's topology condition. Then we announce the new rule with its new topology condition.

Computing route reachability under failures. When the route propagation converges, it is easy to see whether a subnet exists in the RIB of a device. Because every RIB rule in Hoyan has a topology condition, operators can also check whether there exists a failure case that makes routes unreachable to a device under k failure links. If there exists, we can conclude that the reachability is not resilient with up to k link failures. Therefore, given the candidate rules $r_1, \ldots, r_n$ and their topology conditions $R(r_1), \ldots, R(r_n)$, the topology condition which makes at least one rule exist is $V = R(r_1) \vee \ldots \vee R(r_n)$. By leveraging logic solver (*e.g.*, Z3 [10]), we can easily get the minimum number of False variables to make V be False [2, 3, 32]. For instance, in Figure 4, the topology condition for D to receive at least one route to subnet N is $V = (a_1 \wedge a_4) \vee (\neg a_1 \wedge a_2 \wedge a_3 \wedge a_4)$. We can see that if a_4 is False, V will be False.

Algorithm 1 in Appendix A details the workflow of the above route reachability simulation.

Handling route update racing. Similarly to the failure handling, we can also create logical formulas in route selections to judge whether there is ambiguity in route convergence, which is the root cause of vulnerability to non-deterministic route arrival orders. For instance, in the example of Figure 1(c), for each IP prefix, we propagate all of its route updates without dropping due to route selection. After the simulation, each router will receive all routes that ingress and egress filter policies permits. In this case, A receives routes $m_{D\to B\to A}$ and $m_{C\to A}$, and B receives routes $m_{C\to A\to B}$ and $m_{D\to B}$. A puts $m_{D\to B\to A}$ in higher priority because it has a higher local preference, while B prefers $m_{C\to A\to B}$ because A enlarges its weight at the egress port. We can encode the logical relationship of the route selection process with a group of formulas. If we can find more than one solution to the formulas, it means the route convergence is ambiguous, and the configuration is buggy under route update racing (see Appendix B for more details).

One might argue that in the worst case the route propagations without route selection drops can be intractable in large networks. However, in practice, since we have ingress and egress policies to filter routes, the actual number of potential propagation paths for a single IP prefix is moderate.

Supporting iBGP, IS-IS/OSPF, and static routes. By far the route propagation process is designed for path-vector protocols like eBGP. For iBGP, the existence of a peering session depends on whether the routes of peers are reachable to each other. The reachability of iBGP peers is provided by IS-IS protocol on our WAN. Therefore, Hoyan first computes routes from IS-IS (OSPF follows the same process),

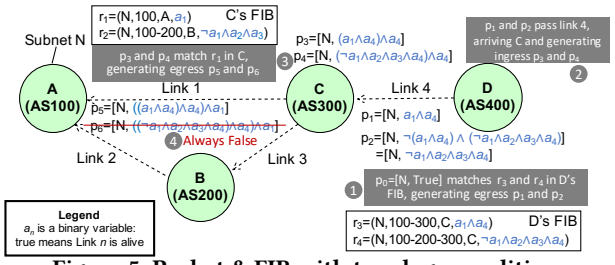


Figure 5: Packet & FIB with topology conditions.

then decides the peering sessions of iBGP, and finally propagates BGP routes over eBGP and iBGP sessions. To compute the topology conditions of IS-IS, we transfer IS-IS into a path-vector protocol and make sure each node selects routes based on the weighted shortest path. The weights on links are consistent with IS-IS configurations. Based on our more-than-one-year experience, the final converged RIBs from the converted protocol are identical to original IS-IS. Therefore, the RIBs from IS-IS has topology condition encoded. The topology condition of an iBGP session is a combination of the topology conditions of the IS-IS routes the iBGP session uses, and routes propagated over an iBGP session keeps the topology condition of the iBGP session. Incidentally, static routes are added to their original routers' RIBs without any initial topology conditions. Please see Appendix C for more details about IS-IS and iBGP supports.

5.5 The reachability of packets

Our operators also need to reason about the reachability of a packet with a destination from a group of network devices to the gateway of the destination subnet. Although route reachability does not mean packet reachability, route reachability to a destination is the necessary condition of packet reachability. Specifically, after the route reachability simulation, Hoyan gets RIBs on each device in which each rule has its own topology condition, and it merges the RIBs from different protocols and static routes into FIBs. Rules in FIBs inherent and combine the topology conditions of their corresponding RIB rules. To exam the reachability of a packet, Hoyan makes a symbolic execution over the FIBs and drops the packet if the topology condition is False under up to k failures. Finally, if the packet can reach the destination, it means "reachable" under up to k failures. Figure 5 illustrates the process of packet reachability verification from D to A following the example of Figure 4. Please see Appendix D for more details about the reachability of packets.

5.6 Optimizations for scalability

One potential concern is that tracing topology conditions in the route or packet propagation could grow exponentially, due to the logic implications of multiple paths, route aggregations, or ECMP. However, we take the following strategies that make Hoyan scale substantially. Section 8.1 shows the effectiveness of these optimizations at the scale of our full WAN.

Dropping more-than-k-failure conditions. If the topology condition of a route has already contained more than k negation of link aliveness, the route will be dropped because we only care about the failure cases with no more than k link failures. Since k is small (*e.g.*, k=3) compared with the total number of links, this optimization significantly reduces the number of branches to explore.

Dropping impossible conditions. It is also easy to judge whether a formula is always False for further pruning. For example, in Figure 5, we stop considering p_6 at step ④ because its topology condition is always False.

Simplifying condition formulas. There are at most the number of links variables in a topology condition formula. In addition, a route update or a packet usually only passes through a small number of links. Hence, the number of independent variables in a topology condition is typically small, so that a topology condition can be simplified to a short formula. We can achieve great memory efficiency from such simplification.

Another concern in the pruning strategies is the impact of "late high ranked rules". Specifically, if we announce a low ranked rule first and make the amendment to its topology condition later, whether the pruning decisions we have already made before are still valid. Fortunately, the answer is yes: suppose r is the low ranked rule, and $R(r, D)$ is its old topology condition when it is first announced. Later, after the high ranked route r' comes, r's topology condition will be modified to $R(r, D) \wedge \neg R(r', D)$. If r has already been dropped at D, $R(r, D)$ has either more than k negative variables or is always False. Modifying $R(r, D)$ to $R(r, D) \wedge \neg R(r', D)$ can neither reduce the number of negative variables nor make $R(r, D) \wedge \neg R(r', D)$ be True (if $R(r, D)$ is False). Therefore, the pruning decisions remain valid after topology condition amendments.

6 BEHAVIOR MODEL TUNER

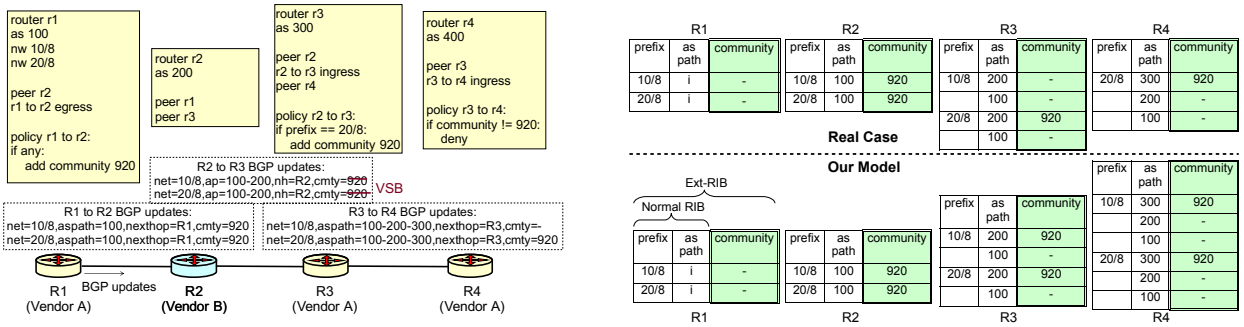
In §5, we see the accuracy of network models serves as the foundation of verification correctness: even a small logic flaw in the network model is likely to cause incorrect results (see §7). This section describes how Hoyan uses its model tuner to continuously detect flaws caused by VSBs and offers high fidelity verification.

To build a model tuner, we employ a black-box testing approach to compare the behaviors of our model with real devices, as vendors typically do not share the details of their implementations. The basic idea to find flaws in current device behavior models is to compare whether the model can have the same output with the same input as a real device in different network environments. Nevertheless, there are several critical challenges to realize this simple idea:

Unpredictable VSB areas. Despite that our operators may know some VSBs based on their experience, we still need to broadly check the behavior models and detect new VSBs. Hence, we aim to create various environments to check our models rather than constructing several special cases for particular VSBs. We also try to localize the root cause if a mismatch between our model and real devices is found rather than assuming it is caused by any known VSBs.

Coverage of comparison cases. A device needs a context to perform a behavior. For instance, a BGP router needs to receive proper routes to perform route aggregation. Or it needs to get updates with private ASes in the path to perform `remove-private-AS`. Even for a single type of device and a single route protocol, there are too many cases to cover in general to fully validate whether a device behavior model matches the actual behavior of real devices.

Our pragmatic strategy to address the coverage issue is to make sure we first cover all cases that a device faces in production. Therefore, the behavior model validator (Figure 2) keeps monitoring the online configuration and the route propagation of the production



(a) Routers and their configurations.

(b) Comparing ext-RIB in real and our model.

Figure 6: A simplified real example of a latent VSB: Vendor A does not remove the community value from its BGP updates by default, while Vendor B does. Our model follows Vendor A's behavior.

network and continuously compares the route propagation it computes from the current behavior model with the real network. Alerts are raised if a difference is detected. Then the behavior model validator localizes the root cause of this difference. We observe that this strategy works very well in practice.

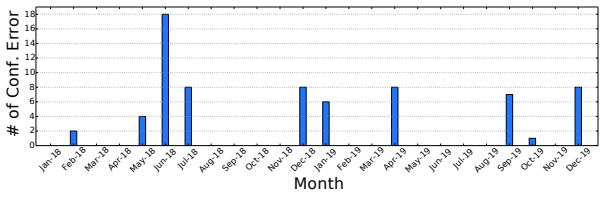
Localization of root causes. First, it is possible to localize root causes at a wrong place if we merely use existing monitoring methods and data to detect mismatches. Because a VSB's impact might only be observed far from the real root cause place. For instance, in Figure 6(a), R2 (Vendor B) by default removes communities from the BGP updates it sends, while other routers (Vendor A) do not. Figure 6(b) shows the RIBs of the four routers. Prefix 10/8 in the RIBs of both R3 and R4 do not match the real case, but R2's is identical. Naturally, one might think the root cause is R3 or R4. However, it is R2, which can only be discovered if we look at the community of each route in R2 and R3's RIBs. In order to accurately locate the root cause, we create a concept called "extended RIB (ext-RIB)" which includes all attributes of each route that can make impacts in route selection. We compare ext-RIBs rather than real RIBs directly from devices for root cause localization.

Second, even with ext-RIBs, some VSBs are still latent. For example, in Figure 6(b), ext-RIBs for prefix 20/8 are identical in all routers. If 10/8 does not exist, the only way to detect the VSB is in the route updates from R2 to R3 (Figure 6(a)); thus, besides ext-RIB, we also collect route updates received by the routers and use BGP monitoring protocol [26] to check the process details. This enables the tuner to precisely locate VSBs between ingress policy and route selector. After that, our operators write patches embedded in corresponding device behavior models to make them accurate. All of these methodologies are necessary to accurately locate the root causes, since VSBs are prevalent in all steps of route processing.

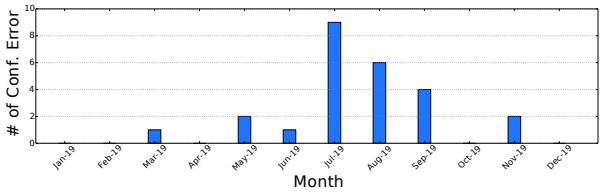
Scalability of model validation. Comparing all IP prefixes' propagation process is not traceable in our network. We split configurations into blocks that each presents a single policy or behavior. We then build an automatic way to suggest a moderate number of prefixes that can cover most configuration blocks, similar to the "equivalent class" idea in ATPG [31].

7 DEPLOYMENT EXPERIENCE

Hoyan has been deployed in production and used on a daily basis on our WAN. Figure 7 presents the monthly configuration error it



(a) Online configuration auditing (Jan 2018 – Dec 2019).



(b) Configuration update validation (Jan 2019 – Dec 2019).

Figure 7: # of Errors found by Hoyan in production.

finds for two-year online configuration audits (Figure 7(a)) and one-year configuration update validations (Figure 7(b)). Also, Table 2 summarizes the realistic VSBs in Table 2 that boosted Hoyan's accuracy from < 10% to > 99%. The bursty phenomena in Figure 7 correlates to our internal network configuration updates, which were mainly caused by our business events such as multiple services upgrading and new cluster launching.

To specific, we present two real cases for preventing uncertainties before updates in §7.1, two real cases for online configuration audits in §7.2, and real detected VSBs in §7.3.

7.1 Detecting uncertainties before updates

This section presents two real-world "detecting update errors" examples: (1) preventing outages before updates, and (2) avoiding impacts of route update racing ahead of time.

Preventing outages before updates. In one of the network upgrades, our operators aimed to change the static-route preference on all the provider-edge (PE) routers – one of the most important roles of routers on our WAN – from 1 to 150. However, there were two old PE routers whose eBGP preferences were configured as 30 due to the specific business reason. Before the upgrade, the static route worked smoothly because its preference is higher, *i.e.*, 1 > 30,

VSB	Description	Affected dev.	# patch-lines
default ACL	Whether permitting data packets that match no explicit ACL.	87.5%	40
default route policy	Whether accepting route updates that match no explicit policy.	82.83%	39
(ext) community	Whether including (extended) communities in route updates by default.	63.91%	46
route redistribution	Whether redistributing default route (0.0.0.0/0).	13.26%	30
AS loop	Whether allowing AS number repetitions in AS Path.	8.63%	26
remove private AS	Whether removing all private ASes from AS path.	7.38%	66
self-next-hop	Whether using self as next-hop when announcing iBGP updates to VPN peers.	6.52%	13
local AS	Whether adding new AS into AS path during AS migration.	1.32%	17

Table 2: Detected VSBs and their impacts.

but if the updated configurations are committed, the eBGP preference would become higher, *i.e.*, 30 > 150, thus blocking all the static routes of those two routers from being activated. It is hard for our operators to notice this error ahead of time because they needed to update hundreds of PE routers but only two of them had the above risk. Fortunately, before our operators committed this update, they checked whether this update met the intended reachability property by Hoyan. Hoyan accurately simulated routes in the updated network and pointed out the potential violations.

Route update racing. Hoyan can also detect potential configuration errors caused by non-deterministic BGP update racing. Figure 1 is a real example that occurred in Alibaba WAN. Specifically, our operators produced a route update plan, where multiple nodes announced a prefix 10.0.1.0/24. Unbeknownst to the operators, if router B receives the BGP update from A, as shown in Figure 1(a), then B would select A as the next-hop rather than D; on the contrary, suppose before A sends the BGP update to B, a BGP update has been sent from B to A, as shown in Figure 1(b). B would become the next-hop of A. Such a misconfiguration is very tricky, because it may lead to multiple routing cases depending on arrival orders of routes. Most of the existing verification systems, *e.g.*, Batfish, cannot detect this type of errors. With Hoyan, our operators detect this update plan may potentially violate our reachability intent ahead of time, preventing a severe service outage.

7.2 Online configuration audits

We present two real examples: (1) online configuration audits, and (2) k-failure tolerance audits.

IP address conflict resulting from misconfiguration. IP address conflicts typically lead to serious service disruptions, because the traffic that should have been sent to router A was actually sent to router B, making A unreachable. Given the complexity of our WAN and the diversity of our hosted businesses, it is much more challenging for the operators to avoid IP address conflicts on our WAN. In one of our WAN expansion events, our operators configured IP addresses for many PE routers. Due to the misunderstanding of the IP address recovery information, our operators configured one of PE routers P with an IP address that had been assigned to a metropolitan-area router M. Coincidentally, they did not import traffic immediately to the PEs; thus, nobody noticed this misconfiguration. However, once our operators import traffic to the PE router P, the traffic would be forwarded to M and immediately crash M and its services, since the metropolitan-area routers, on our WAN, can carry much less traffic than the PE router. Fortunately, Hoyan periodically audits the propagation scope of some critical IP addresses. We found this conflicted IP appeared in some downstream routers that should have it blocked by BGP filters. Hoyan successfully detected this risk before it results in serious outages.

Auditing k-failure tolerance. Our WAN relies on the redundant routers within the same BGP group to avoid common-mode failures; thus, correctly configuring these BGP peers as *equivalent roles* presents an important property for our WAN. Equivalent role property means routers in the same device group should always receive the same route updates and have the same network state. However, due to the high-churn network updates, many BGP peers' configurations are frequently changed to meet diverse business needs; before Hoyan was developed, our operators manually check whether a multitude of updates is correctly configured without introducing any side effects. Note that the misconfigured equivalent role property does not lead to the service disruption immediately, but it introduces potential single-point failure risks to our WAN. Once the bottlenecked links or routers are down, it would result in cascading failures across the entire network. We used Hoyan to proactively detect many configuration errors violating the equivalent roles by running k-failure (*e.g.*, $k = 1$ and $k = 2$) verification on our WAN. The majority of these errors were side-effects introduced by the daily WAN updates. Our experience also indicates that MAN routers (*i.e.*, the edge routers connecting WAN and DCNs) account for the largest percentage of violating the equivalent role property.

7.3 Real-world vendor-specific behaviors

Table 2 lists major VSBs Hoyan found in production which significantly influences the accuracy of verification results. The "Affected dev." column shows the fraction of devices that potentially have the corresponding VSBs, and "# patch-lines" means the lines of code to patch the behavior model. Our behavior model is designed to be highly modulized to embrace VSBs. In most cases, for a new discovered VSB, the operators can write a new patch in $O(10)$ minutes to eliminate future impacts. We pick some example VSBs from Table 2 to explain their impact.

Default policy. "default ACL" and "default route-policy" affect the most of devices on our WAN. Both of them refer to the vendor's default action (permit or deny). If a route update (or packet) does not match any explicit route-policy or ACL rule in a device's configuration, permitting or denying this route update (or packet) depends on the default implementation of this device vendor.

The community in BGP updates. The VSB example in Figure 6 is the "(ext) community" VSB. Some vendors drop the extended communities at default, while others keep. Unaware of this type of VSB, a verification tool may generate a RIB different from the real-world one, like R4's RIB in Figure 6.

AS migration. The configuration of "local AS" is designed to change a router's AS number (*i.e.*, AS migration). At the beginning of an AS migration, operators typically want to keep the old AS number to a router's existing peers for maintaining the BGP

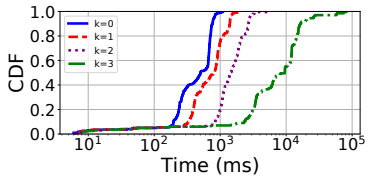


Figure 8: Time to simulate one IP prefix with different k.

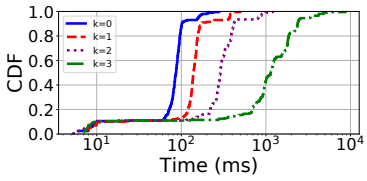


Figure 9: Time to verify one IP prefix with system overhead.

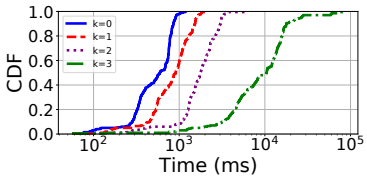


Figure 10: Turnaround time to verify one IP prefix.

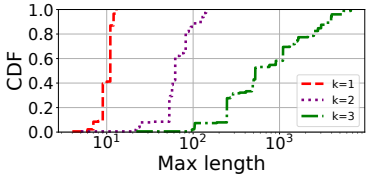


Figure 11: Max length of the topology condition formula of each prefix.

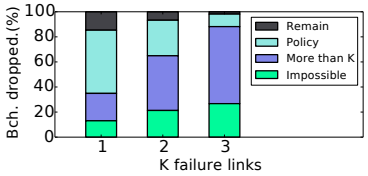


Figure 12: Effectiveness of pruning with different k.

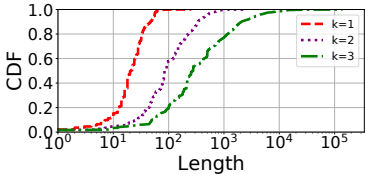


Figure 13: Formula length for reachability checking per prefix.

Network properties		Route reachability	Packet reachability
Reachability	$k = 0$	$481s$	$245s$
	$k = 1$	$770s$	$304s$
	$k = 2$	$1523s$	$715s$
	$k = 3$	$10496s$	$3989s$
Role equivalence		$13s$	-
Route update racing		$3800 - 4400s$	-

Table 3: Time to verify our entire WAN with Hoyan.

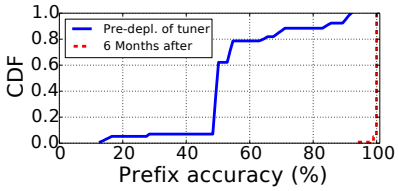


Figure 14: Verification accuracy tuning.

session. They configure the old AS number as "local AS". In the BGP updates of the router under migration, some vendors just put the old AS number in the BGP path, but others put both new and old. This affects the length of AS path which is used in many routers as a metric to select the best route. Therefore, this VSB ultimately affects the reachability decision when verifying an AS migration.

iBGP peering via VPN. If a router A_1 learns a route with next-hop B from its iBGP peer A_2, B would become the next-hop for one of the items in A_1's RIB. However, due to automatically-enabled "self-next-hop" VSB in some vendor, A_1 may learn A_2 as the next hop instead of B. Such a VSB directly causes our simulated model to be inaccurate.

8 PERFORMANCE EVALUATION

In this section, we present some key performance numbers directly from the deployed system to unveil the real running status of Hoyan. We also compare the verification performance of Hoyan with three state-of-the-art verification tools in conducted experiments. We run all evaluations on a server with Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GHz (32 logical cores), 256GB RAM and 1T SSD. Note that Hoyan could be run in a distributed way to get better performance, but our experience shows one machine should be enough.

8.1 Hoyan's performance in the wild

Verification performance. Figure 8 shows the CDF of the time to simulate the propagation process of one IP prefix in Hoyan over our entire WAN. Specifically, 98% IP prefixes can be done within one second. When k becomes large (*e.g.*, $k = 3$), the 90% time cost increases to around 17 seconds.

Once the simulation is done for all prefixes, operators can verify whether network properties are held under normal and failure cases by solving the encoded topology conditions. Figure 9 shows the CDF of the time to verify route reachability queries by the solver. It includes the time on verifying the logical formula by the solver and system overhead (*e.g.*, RPC, data loading, and parsing time). We observe Hoyan can answer these queries in a quite light-weight way after the initial route propagation. The most complicated query under the failure case ($k = 3$) is answered within 8 seconds. Figure 10 shows the end-to-end latency for verifying one IP prefix. Even for $k = 3$ the median is less than 10 seconds.

Pruning (§5) keeps the logic formula simple in the topology encoding. The length of the topology condition formula can good evaluate the complexity, since it indicates how many link-aliveness determines the route reachability. As shown in Figure 11, the maximum topology length in Hoyan was $O(10,000)$ during the route propagation process. Moreover, as shown in Figure 12, when $k = 3$, on average, only 2% conditions survive during the propagation process. 61%, 27% and 10% of conditions are cut due to larger-than-k, impossible conditions and policies respectively. Finally, once the simulation is complete, Figure 13 shows the formula length that Hoyan feeds into the solver for the end-to-end verification result of every prefix. As we can see, the longest formula is only 137,078.

Besides the performance evaluation for individual IP prefixes, we also show the overall performance in real-life verification scenarios. We evaluated the end-to-end verification latency in two scenarios: (i) route reachability verification with network operators' expectations and packet reachability verification for all pairs of devices; (ii) equivalence verification of two devices. These two scenarios verified two properties (route reachability under k-failure and device equivalence) that our operators care the most before pushing network updates to the devices in the real world. Hoyan took 30 seconds on average to load all topology and configuration information. Table 3 shows it can finish all-pair packet reachability verification around 4 hours even with $k = 3$. The device equivalence verification is super fast, which is about 13 seconds on average.

Verification accuracy. We measured the verification accuracy by comparing Hoyan's outputs with the operator's expectation and figuring out the wrong side if a mismatch happens. We define "accuracy" as x/y, where x is # of simulation results that are the same as the actual routing tables, and y is # of simulation/verification results. Figure 14 shows the CDF of prefix accuracy in our WAN. We observe that before we deploy the behavior model tuner, the accuracy of verification is fairly low: 50% of the prefixes have 50% accuracy or lower. The verification accuracy was significantly boosted six months later. After Hoyan discovered and fixed many VSBs, 95% of our prefixes have reached 100% accuracy. The remaining inaccuracy is caused by incomplete data input fed by the external system, rather than the quality of Hoyan's behavior model.

Model tuner performance. We also evaluated model tuner. Please see Appendix E for more details.

8.2 Comparing Hoyan with existing tools

We compared Hoyan with Batfish [12], Minesweeper [4], and Plankton [24] on verification performance with conducted experiments.

Due to the scalability issue, the alternatives cannot verify our entire WAN with failure coverage. We thus picked two subnets from our WAN – a small one with 20 routers and a medium one with 80 routers – and used these three tools to verify them for performance comparison.

We evaluated two types of reachability properties. The first is k-failure packet reachability. We verified packet reachability between each pair of routers in the network. We used 50 threads to perform verification of different device pairs in parallel for all three approaches. The second is device equivalence. We verified if two selected routers build the same RIBs. We used a single thread in this scenario. Due to limited space, we put detailed comparison results and discussions in Appendix F.

Packet reachability. In the small subnet, Hoyan and Batfish took 3 and 28 seconds to finish the verification, respectively; however, Minesweeper spent 1555 seconds. When $k = 3$, however, Batfish and Plankton have timed out, and Minesweeper took more than two hours; on the contrary, Hoyan only needed 14 seconds. In the medium subnet, Hoyan performed orders of magnitude efficient than all alternatives (see Appendix F for details).

Device equivalence. Given two devices, Hoyan can verify whether they receive the same route updates and build the same RIB/FIBs. In both small- and medium-subnets, Hoyan took less than 4 seconds to finish the verification, whereas Minesweeper needed 203 seconds and more than one day respectively. We did not evaluate Batfish or Plankton since their code does not have this feature.

9 DISCUSSION AND LESSONS

The accuracy and correctness of Hoyan. In principle, as a network configuration verifier, Hoyan's goal is to check whether the configuration of interest meets the intent expressed by the operators; thus, Hoyan's correctness means there is, compared with the actual cases (*e.g.*, RIBs and packet reachability), no false positive or false negative in Hoyan's reasoning results. While Hoyan's verification algorithms are not proved in a mathematical way, but based on our recent-one year experience Hoyan has never performed false positives or false negatives.

Lessons and practical issues. Besides VSBs, another major practical issue influencing the accuracy is the lack of high fidelity configuration parsers to merge incremental updates with an existing snapshot of configurations. Existing solutions need a complete configuration snapshot for verification. However, what operators write are incremental command lines into devices. It is hard for operators to manually and correctly generate the complete configuration of each device for update verifications. On the other hand, automatically generating configuration snapshot is difficult because it is also arduous to write configuration parsers for all types of devices and make sure they behave in the same way as real devices. We have developed many templates for automatically mapping the "operator-input" incremental command lines to the complete configuration "fed" into Hoyan. It spent too much time and many efforts on developing these templates.

In addition, it is also hard to precisely know the all detailed reachability intent/properties on such a large scale WAN which carries multiple businesses that have complex reachability relationships among each other. It is encouraging to see that this problem has drawn attention to the community [6, 8, 9, 17, 30], and we would try these ideas in Hoyan in the future.

Can verified router implementation avoid VSBs? The key reason for VSBs, in fact, is non-transparency of commercial routers, rather than whether the router implementation is verified or not. Because it is hard for us to know what default behaviors the router vendors implemented, VSBs can always exist in the future. This is also the reason we need to develop a timely model tuner to proactively detect VSBs hidden in routers. However, if some router vendor can fully verify their router implementation and release their behavior specifications, that would be definitely helpful to avoid VSBs, because Hoyan can build its behavior model by just following the vendor released specifications.

10 CONCLUSION

This paper presents Hoyan, the first-ever reported practical configuration verification system deployed on Alibaba's global-scale WAN. We introduce the challenges on accuracy, scalability, and coverage to build a configuration verifier in production WAN. Hoyan's innovation has two folds First, it is the first configuration verifier that considers VSBs and achieve near-100% accuracy in the production environment; Second, it has good scalability to verify networks under uncertainties, thanks to its novel "global simulation & local formal-modeling" design. Our operators have used it on a daily basis for two years.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We thank our shepherd, Olivier Bonaventure, and SIGCOMM reviewers for their insightful comments. We also thank Yahui Li for her efforts on the earlier version of this work. Fangdan Ye and Zhiliang Wang are with Institute for Network Sciences and Cyberspace of Tsinghua University and Beijing National Research Center for Information Science and Technology. Fangdan Ye and Zhiliang Wang are supported in part by the National Key R&D Program of China 2018YFB1800205. This work was supported by Alibaba Group through Alibaba Innovative Research Program.

REFERENCES

[1] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Tiramisu: Fast and General Network Verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[2] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. 2009. Solving (weighted) partial MaxSAT through Satisfiability Testing. In *12th International Conference on Theory and Applications of Satisfiability Testing (SAT)*.

[3] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. 2010. A new algorithm for weighted partial MaxSAT. In *24th Conference on Artificial Intelligence (AAAI)*.

[4] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A general approach to network configuration verification. In *ACM SIGCOMM (SIGCOMM)*.

[5] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2018. Control plane compression. In *ACM SIGCOMM (SIGCOMM)*.

[6] Ryan Beckett and Ratul Mahajan. 2019. Putting network verification to good use. In *18th ACM Workshop on Hot Topics in Networks (HotNets)*.

[7] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2016. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *ACM SIGCOMM (SIGCOMM)*.

[8] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin T. Vechev. 2018. Net2Text: Query-guided summarization of network forwarding behaviors. In *15th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[9] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin T. Vechev. 2020. Config2Spec: Mining network specifications from network configurations. In *17th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[10] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[11] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient network reachability analysis using a succinct control plane representation. In *12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*.

[12] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A general approach to network configuration analysis. In *12th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[13] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast control plane analysis using an abstract representation. In *ACM SIGCOMM (SIGCOMM)*.

[14] Alex Horn, Ali Kheradmand, and Mukul R. Prasad. 2017. Delta-net: Real-time network verification using atoms. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[15] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C. Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, and Parag Sharma. 2019. Validating datacenters at scale. In *ACM SIGCOMM (SIGCOMM)*.

[16] Jesper Stenbjerg Jensen, Troels Beck Krøgh, Jonas Sand Madsen, Stefan Schmid, Jiří Srba, and Marc Tom Thorgersen. 2018. P-Rex: Fast verification of MPLS networks with multiple link failures. In *14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*.

[17] Siva Kesava Reddy K., Alan Tang, Ryan Beckett, Karthick Jayaraman, Todd D. Millstein, Yuval Tamir, and George Varghese. 2020. Finding network misconfigurations by automatic template inference. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[18] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header space analysis: Static checking for networks. In *9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[19] Ahmed Khurshid, Xuan Zhou, Whenxuan Zhou, Matthew Caesar, and Philip Brighten Godfrey. 2013. VeriFlow: Verifying network-wide invariants in real time. In *10th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[20] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. 2017. CrystalNet: Faithfully emulating large production networks. In *26th Symposium on Operating Systems Principles (SOSP)*.

[21] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking beliefs in dynamic networks. In *12th USENIX Symposium on Networked System Design and Implementation (NSDI)*.

[22] Nuno P. Lopes and Andrey Rybalchenko. 2019. Fast BGP Simulation of Large Datacenters. In *20th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*.

[23] Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. 2016. Scaling network verification using symmetry and surgery. In *43rd ACM Symposium on Principles of Programming Languages (POPL)*.

[24] Santhosh Prabhu, Kuan Yen Chou, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. 2020. Plankton: Scalable network configuration verification through model checking. In *17th USENIX Symposium on Networked System Design and Implementation (NSDI)*.

[25] Bruno Quoitin and Steve Uhlig. 2005. Modeling the routing of an autonomous system with C-BGP. *IEEE Network* 19, 6 (2005), 12–19.

[26] J. Scudder, R. Fernando, and S. Stuart. 2016. *BGP Monitoring Protocol (BMP)*. RFC 7854. IETF. http://tools.ietf.org/rfc/rfc7854.txt

[27] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. 2016. SymNet: Scalable symbolic execution for modern networks. In *ACM SIGCOMM (SIGCOMM)*.

[28] Bingchuan Tian, Xinyi Zhang, Ennan Zhai, Hongqiang Harry Liu, Qiaobo Ye, Chunsheng Wang, Xin Wu, Zhiming Ji, Yihong Sang, Ming Zhang, Da Yu, Chen Tian, Haitao Zheng, and Ben Y. Zhao. 2019. Safely and automatically updating in-network ACL configurations with intent language. In *ACM SIGCOMM (SIGCOMM)*.

[29] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. 2016. 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)*.

[30] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. 2019. dShark: A general, easy to program and scalable framework for analyzing in-network packet traces. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[31] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Automatic test packet generation. In *8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*.

[32] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. 2020. Check before you change: Preventing correlated failures in service updates. In *17th USENIX Symposium on Networked System Design and Implementation (NSDI)*.

APPENDIX

Appendices are supporting material that has not been peer reviewed.

A ROUTE REACHABILITY REASONING

Algorithm 1 details our route reachability reasoning (§5.4) for a given prefix. The workflow in Algorithm 1 exactly corresponds to what we describe in §5.4. We want to emphasize the topology condition encoding occurs in Line 20 in Algorithm 1, *i.e.*, the `update_condition()` function, and the "late higher priority routes" case is handled by the withdraw function (Line 24-32 in Algorithm 1), *i.e.*, the `withdraw()` function.

In addition, Line 21 in Algorithm 1 builds a propagation tree, thus enabling us to track all the route messages via this tree. In the `withdraw()` function, for a route message to be withdrawn, m, its child nodes would be deleted from m's RIBs, as shown in Line 26 in Algorithm 1, and such removal would lead to cascading effects to the lower-priority items in these child nodes, as shown in Lines 27-31 in Algorithm 1.

B HANDLING ROUTE UPDATE RACING

Non-deterministic route update racing is a tough problem during the process of route convergence. Figure 1 has shown an example our operators met in practice. Similarly to the approach for the "route reachability under failures" case, Hoyan creates logical formulas in route selections to judge whether there is ambiguity in route convergence, which is the root cause of non-deterministic route arrival orders. The algorithm processes as the following steps. (i) For each IP prefix p, we simulate its route update propagations without taking into account any *route deny* policy.

Algorithm 1: Route reachability for a given prefix

```
   Input  : A is the announcement node of given prefix
   Input  : Q is an empty queue
1  Function RouteReachability():
2      foreach p ∈ the peer set of A do
3          Q.Push(m_{A→p});
4      while Q is not empty do
5          m_{i→j} ← Q.Pop();
6          if j.check_ingress(m_{i→j}) = DENY then
7              continue;
8          j.route_selection(m_{i→j});
9          j.RIB ← j.update_RIB(m_{i→j});
10         S = ∅;
11         /* Loop below handles the late higher priority case */
12         foreach d ∈ j.RIB do
13             if d.priority is lower than i then
14                 S.Push(d);
15                 d.withdraw(Q);
16         S.Push(i);  // If i is the lowest priority, S only contains i
17         foreach k ∈ the peer set of j do
18             foreach d ∈ S do
19                 if j.check_egress(m_{d→j}) = ACCEPT then
20                     m_{j→k}.update_condition(r_{d→j} ∧ a_{j→k});
21                     m_{i→j}.son_set.add(m_{j→k}); // Propagation tree
22                     Q.Push(m_{j→k});
23     /* By far, all the node's RIBs for the given prefix have been simulated */
   Input  : Route Message m_{i→j}
24 Function withdraw():
25     foreach son_{j→k} ∈ the son set of m_{i→j} do
26         k.delete_from_RIB(son_{j→k});
27         foreach f ∈ k.RIB do
28             if f.priority is lower than son_{j→k} then
29                 f.update_condition();
30                 Q.Push(f);
31                 f.withdraw(Q);
32         son_{j→k}.withdraw(Q);
```

(ii) Until the above simulation for p finishes, each router receives p's route updates, generating its RIB based on its own route selection policy. For example in Figure 1(c), router A receives routes $m_{D\to B\to A}$ and $m_{C\to A}$, and B receives routes $m_{C\to A\to B}$ and $m_{D\to B}$. A puts $m_{D\to B\to A}$ in higher priority in its RIB because it has a higher local preference, while B prefers $m_{C\to A\to B}$ in B's RIB because $m_{C\to A\to B}$'s weight is enlarged when it passes through A's egress port.
(iii) We encode the logical relationship of the route selection process for each router via exactly same way as the topology condition deriving process mentioned earlier. For example in Figure 1(c), router A's RIB is encoded as $I_{D\to B\to A} \vee I_{C\to A}$, and router B's RIB is encoded as $I_{C\to A\to B} \vee I_{D\to B}$.
(iv) We finally get a big formula by connecting all routers' formulas with ANDs, and solve this big formula via SMT solver. If we can find more than one solution from the solver, it means the route convergence is ambiguous, and the configuration can lead to non-deterministic route update racing. For example in Figure 1(c), the big formula is $I_{D\to B\to A} \vee I_{C\to A} \wedge I_{C\to A\to B} \vee I_{D\to B}$. If we solve this formula, we get two solutions: (1) $I_{C\to A\to B} = I_{C\to A} = \mathit{True}$ and $I_{D\to B} = I_{D\to B\to A} = \mathit{False}$; and (2) $I_{C\to A\to B} = I_{C\to A} = \mathit{False}$ and $I_{D\to B} = I_{D\to B\to A} = \mathit{True}$. These two solutions exactly correspond to the cases shown in Figure 1(a) and (b), respectively.

C SUPPORTING IS-IS AND IBGP

In general, we address IS-IS reachability verification by reducing it to the problem of checking BGP reachability (detailed in §5.4). Algorithm 2 shows the entire workflow of reasoning about the IS-IS route reachability.

In particular, within the same Level 1 (or L1), our key idea is to translate each IS-IS node into a BGP node with weight attribute, and then leverage the Shortest Path First (SPF) algorithm to compute the optimal paths, thus getting the IS-IS routing table for each IS-IS node. Such a design enables us to directly use the topology condition encoding approach (§5.4) to verify the reachability properties (*e.g.*, k-failure tolerance) for IS-IS protocol.

Mapping an IS-IS node to a BGP node. We translate an IS-IS protocol network into a BGP network model (*e.g.*, Figure 4) based on the following two steps:
(i) Each IS-IS node in our network model is constructed the same as a BGP node (*i.e.*, the device node in §5.4). But each IS-IS node just has one more attribute than a BGP node, the transitive IS-IS weight attribute (called ISIS-weight). ISIS-weight has a higher priority than the AS number attribute in the route selection process. For ISIS-weight, its default value is zero, the smaller the value, the higher the priority.
(ii) The neighbor of each IS-IS node is the same as the one in the BGP case. The weights of links between IS-IS neighbors are stored as the egress policy of BGP peers. In this way, each IS-IS node sending an IS-IS route increases its weight by its corresponding IS-IS weight.

After the above transformation of IS-IS nodes and links, we obtain a BGP network model that represents our current IS-IS network reachability. We can then use the topology condition encoding approach (detailed in §5.4) to check the properties of interest in the original IS-IS network.

As shown in Algorithm 2, the propagation process of IS-IS protocol in Hoyan is quite similar to how Hoyan simulates the propagation of BGP routes. The key difference is Line 13 in Algorithm 2 increases the weight of messages because the new route selection function in IS-IS model needs to select the best route based on these weights. In other words, there are two differences between `RouteReachability` and `RouteISISReachability`: (1) Line 26 increases the weight of messages during the propagation, and (2) Line 15 is the new route selection function which selects routes by taking into account IS-IS weights. The above design enables us to verify the k-failure tolerance property in IS-IS network model by the approach similar to the topology condition encoding in §5.4.

Route redistribution simulation. Because both topology condition encodings of BGP and IS-IS networks are independent, we obtain two independent routing tables for each node. We thus can get the eventual simulated routing tables by simulating the regular routing distribution between the two tables (*i.e.*, BGP and IS-IS routing tables).

Handling the L1-L2 case. So far, what we did is to simulate IS-IS network within the same L1. Nodes within the same L1 cannot send

Algorithm 2: IS-IS route reachability for a given prefix

```
   Input  : BGP network model B
   Input  : A is the announcement node of given prefix
1  Function MapBGPtoISIS():
2      foreach n ∈ B do
3          Use AS number of n as its unique ID;
4      Add a new route selection policy ISIS-weight in route_selection();
5      Set the priority of ISIS-weight higher than AS path;
6      /* The new route selection is based on the weight*/
       RouteISISReachability();
7      /* By far, all the node's IS-IS RIBs for the given prefix have been
          simulated */
8  Function RouteISISReachability():
9      foreach p ∈ the peer set of A do
10         Q.Push(m_{A→p});
11     while Q is not empty do
12         m_{i→j} ← Q.Pop();
13         if j.check_ingress(m_{i→j}) = DENY then
14             continue;
15         j.route_selection(m_{i→j}); // route selection based on ISIS-weight
16         j.ISIS-RIB ← j.update_RIB(m_{i→j});
17         S = ∅;
18         foreach d ∈ j.ISIS-RIB do
19             if d.priority is lower than i then
20                 S.Push(d);
21                 d.withdraw(Q);
22         S.Push(i);
23         foreach k ∈ the peer set of j do
24             foreach d ∈ S do
25                 if j.check_egress(m_{d→j}) = ACCEPT then
26                     m.isis_weight += get_ISIS_weight(j → k);
27                     m_{j→k} ← r_{d→j} ∧ a_{j→k};
28                     Q.Push(m_{j→k});
```

routes to nodes in other L1. All L1 routes are sent to L2 through the L1/L2 routers, if the route penetration is configured. We thus decide to use the community in BGP to mimic the route penetration in IS-IS. Specifically, in our IS-IS simulation, we use the community to control whether L1 can communicate with L2. As shown in Algorithm 2, because the entire IS-IS route reachability reasoning has been reduced to the problem of BGP route reachability, the reachability between L1 and L2 can perfectly be solved by using community control. In fact, Alibaba network controls L1/L2 communication in IS-IS by binding it with the community in BGP.

Simulating iBGP. In Alibaba, iBGP is run upon IS-IS; thus, iBGP protocol is easily simulated as long as we can support IS-IS protocol. Specifically, the topology condition of an iBGP session is a combination of the topology conditions of the IS-IS routes the iBGP session uses, and routes propagated over an iBGP session keeps the topology condition of the iBGP session. It is straightforward for us to generate the topology conditions for route reflectors (or RRs). RR nodes are just treated as regular BGP nodes with special RR's route selection policies. As shown in Algorithm 2, RR's route selection policies can be added in Line 15, and the entire algorithm workflow does not need to be changed.

D THE REACHABILITY OF PACKETS

Operators also need to judge the reachability of a packet with a destination from a group of network devices to the gateway of the destination subnet. Although, as we mentioned, route reachability does not mean packet reachability, route reachability to a destination is the necessary condition of packet reachability. Hence, the first step to verify packet reachability is to finish the route propagation process and generate FIB from RIB in each device. As we presented in §5.3, each FIB rule takes the topology condition from its corresponding RIB rule. Figure 5 shows the packet propagation process with topology conditions. We omit packets from C to A through B due to space limit. As we can see, every device's FIB inherits its RIB's topology conditions as in Figure 4.

Iteratively deriving topology conditions of packets. Similar to a route update, a packet also has a topology condition to reach the ingress or the egress of a device behavior model's data plane pipeline. The topology condition of packets can also be derived iteratively with two simple rules:

(i) *From FIB to egress*: Suppose $r^1, \ldots, r^n$ are rules that match a packet p's destination, and they are ranked from high priority to low [2]. p will hit the highest ranked rule and get forwarded to the rule's nexthop – if it is an ECMP rule with multiple nexthops, each nexthop will have a copy of p. Suppose $I(p, S)$ is the topology condition for p to enter S's ingress, and $R(r^i, S)$ is the topology condition of r^i for existing in S, the topology condition for p to enter S's egress interface indicated by r^i is $E(p, S)^i = I(p, S) \wedge \neg(R(r^1, S) \vee \ldots \vee R(r^{i-1}, S)) \wedge R(r^i)$. For example, at Step ① in Figure 5, a packet p_0 from D to subnet N can hit two rules in D's FIB, and p_1 and p_2 represent p_0 with different topology conditions by hitting different rules.

(ii) *From egress to otherside's ingress*: For a packet p with topology condition $E(p, S)$ in S's egress, the topology condition for p to reach nexthop D's ingress is $I(p, D) = E(p, S) \wedge a_l$ where l is the link from S to D. For example, at step ② in Figure 5, p_3 and p_4 are generated by this rule.

FIB's topology condition is fixed during the packet propagation, since packets, unlike route updates, cannot change RIB/FIB in routers. Hence, similar to Step ①, p_3 and p_4 hit r_1 in C, generating p_5 and p_6 at step ③. Note that in Hoyan a packet can also be symbolic. The only difference is that topology conditions are attached to each packet branch during symbolic execution.

Computing packet reachability under failures. After the packet propagation process, there can be multiple copies of the packet with different topology conditions reaching the gateway of the destination subnet. Similarly, we can combine all topology conditions and check whether there exists a failure case with less than k failures which eliminates the reachability of the packet.

About equal-cost multi-path routing (ECMP). Current Hoyan does not support ECMP-level packet reachability reasoning due to the following reasons. First, in Alibaba's WAN architecture, ECMP is only configured within the same *device groups*. A device group consists of multiple redundant routers with the same forwarding behaviors. Under such an architectural assumption, the targets of ECMP must be within the same device group, so that the forwarding behavior of any packet should be equivalent. Second, Alibaba

[2] There can be rules with different subnet granularity matching the destination. Based on the longest prefix matching strategy, we first rank rules based on their subnet granularity and put smaller subnet granularity to higher priority. We keep the rank of the rules in the same subnet as what they are in the RIB.

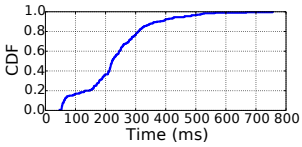


Figure 15: Ext-RIB loading.

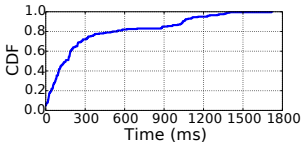


Figure 16: VSB localizing time.

employs another internal system specific to ensure the targets of ECMP are configured correctly within the same device groups. We leave the ECMP reasoning support to the future work.

E PERFORMANCE OF MODEL TUNER

To present the overhead of our behavior model tuner, we pick 200 IP prefixes in our WAN. These prefixes not only cover the most important services of our company, but also are active to serve for millions of users. We measure each prefix in detail to show the performance and overhead of the behavior model tuner.

Figure 15 shows the CDF of the time to load ext-RIB from a real device. In this process, Hoyan needs to contact the target router and pull the network state back. As we can see, the loading time is 222 milliseconds and 382 milliseconds in 50% and 90% percentile respectively. Even for the prefix with the largest propagation scope, the loading time is less than 800 milliseconds.

We measure the memory cost of this process. Loading ext-RIB for one IP prefix from one device is around 1 KB. The total memory cost for all these 200 prefixes is around 840 MB.

After the ext-RIBs are loaded, Hoyan checks and locates VSBs. We evaluate the time cost of this process in Figure 16. In most (90%) cases, Hoyan takes less than 1 second.

F COMPARING WITH EXISTING TOOLS

We compared Hoyan with Batfish [12], Minesweeper [4], and Plankton [24] on verification performance with conducted experiments.

Experiment setup. Because the alternatives cannot verify our entire WAN with failure coverage due to the scalability issue, we pick two subnets from our WAN – a small one with 20 routers and a medium one with 80 routers – and use the three tools to verify them for performance comparison.

We perform two types of reachability verifications. The first is k-failure packet reachability. We verify packet reachability between each pair of routers in the network. We use 50 threads to perform verification of different device pairs in parallel for all three approaches. The second is device equivalence. We verify if two selected routers receive the same routes and build the same RIB. We use a single thread in this scenario.

Network properties		Hoyan	Minesweeper	Batfish	Plankton
Reachability	$k=0$	$3s$	$1555s$	$28s$	-
	$k=1$	$4s$	$3573s$	$6299s$	$50s$
	$k=2$	$5s$	$4733s$	$>24h$	$3369s$
	$k=3$	$14s$	$7430s$	$>24h$	$>24h$
Role equivalence		$3s$	$203s$	-	-

Table 4: Time comparison in the small subnet.

Network properties		Hoyan	Minesweeper	Batfish	Plankton
Reachability	$k=0$	$14s$	$84043s$	$683s$	-
	$k=1$	$22s$	$>24h$	$>24h$	$835s$
	$k=2$	$43s$	$>24h$	$>24h$	$>24h$
	$k=3$	$176s$	$>24h$	$>24h$	$>24h$
Role equivalence		$4s$	$>24h$	-	-

Table 5: Time comparison in the medium subnet.

Packet reachability. Table 4 shows the result in the small subnet. In the $k=1$ case, Hoyan takes 4 seconds to finish the reachability verification whereas Minesweeper, Batfish, and Plankton need 3573 seconds, 6299 seconds, and 50 seconds respectively. When considering failure cases with different k, time cost for Hoyan are increasing slightly to 14 seconds. The time cost for the other two works increases dramatically. For instance, Minesweeper needs thousands of seconds and Batfish takes more than 24 hours when $k=3$.

In the medium subnet, Hoyan performs orders of magnitude faster than all alternatives. Batfish, Minesweeper, and Plankton (when $k \geq 2$) need more than one day under different k as shown in Table 5.

We also evaluate the maximum length of the topology condition formula of each prefix in these two networks. Answering the reachability of a given route, in most cases, Hoyan needs to solve logic formulas of size 242 and 543 when $k=3$ in the small and medium subnets respectively while Minesweeper reaches 230,403 and 4,786,577.

The above evaluation results demonstrate the huge advantage of Hoyan in scalability to verify network configuration with failure cases.

Device equivalence. Given two devices, Hoyan can verify whether they receive the same route updates and build the same RIB/FIB. The equivalence property needs not to consider the failure case since verifying this property is not required to be valid under failures. Table 4 and Table 5 show Hoyan takes less than 4 seconds to verify this property whereas Minesweeper needs 203 seconds and >24 hours respectively. We did not evaluate Batfish or Plankton since their code does not have this feature.