Network Specification Mining with High Fidelity and Scalability

Ning Kang, Peng Zhang, Hao Li, Sisi Wen
Xi'an Jiaotong University

Chaoyang Ji, Yongqiang Yang
Huawei Cloud Computing Technologies Co., Ltd.

2023 IEEE 31st International Conference on Network Protocols (ICNP) | 979-8-3503-0322-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICNP59255.2023.10355598

***Abstract*—Network specification, which describes what an existing network is designed for, can help operators better understand and manage their networks, and is a critical precondition for network verification and synthesis tools to work. Mining specification with existing tools either cannot scale to large networks, or scale better at a cost of sacrificing fidelity. This paper presents *NetMiner*, which can simultaneously achieve high scalability and fidelity. The key idea of *NetMiner* is to use off-the-shelf network simulators to compute routes, and then check properties with data plane verifiers, so as to achieve a high fidelity. At the same time, *NetMiner* improves the scalability by identifying relevant failure scenarios, and aggregating them to significantly reduce the number of needed simulations. This process is solely based on the routes returned by the simulators and therefore preserving fidelity. Experiments using real configurations from a large cloud service provider and synthetic configurations show that *NetMiner* is about 10 times faster than the state-of-the-art.**

I. Introduction

Managing large-scale networks is hard, considering the network configurations are becoming more and more complex. To simplify the management task, network configuration verification and synthesis have been studied in the past decade, which can verify the correctness of existing configurations [1]–[7], generate the configurations from scratch [8]–[13], and update the configurations based on the high-level intents [14]. One critical prerequisite of realizing the above vision is the *network specification*, e.g., reachabilities between the prefixes and ports, waypoints that packets should traverse, and the number of link/node failures that can be tolerated.

Unfortunately, such specification is usually missing or incomplete in practice. The reason is that instead of building a clean-slate fresh network, operators tend to construct the network based on existing configurations. And through the evolution over the years, it is too harsh for the operator to manually compose the specifications. This motivates the problem of *network specification mining*: how to automatically extract the network specification from the existing configurations?

One approach to network specification mining is to simulate or emulate the control plane with control plane simulators or emulators [15]–[18] to generate the routes (RIBs) and forwarding rules (FIBs), and check whether all possible candidate properties (e.g., all-pair reachability) hold with data plane verifiers [19]–[22]. However, such simulation-based approaches suffer from the *scalability* problem when considering the failure tolerance: one needs to simulate the network a lot of times for different failures, and each simulation is considerably costly. We break down this problem into the following two factors. (1) The number of simulations. To extract the complete specification, the verifier needs to simulate *each failure scenario*, and check all properties on it. For example, to verify a set of properties on a network with L links, one has to enumerate all scenarios that any l of L links fail. This results in $O(C_L^l)$ simulations in total, that is, $O(10^8)$ simulations when $l = 3$ and $L = 1000$. (2) The cost of each simulation. To achieve high fidelity, besides modeling the Layer-3 route computation, the simulation must also consider the semantics of other layers. For example, the simulation should model the Layer-2 computation (VLAN, STP, etc.), to account for physical Layer-1 topology. This makes the simulation much more expensive. Considering there are 100 nodes, each with 100 physical/virtual ports, and each port is configured to allow 2000 VLANs, existing simulator will take $O(10^7)$ operations to compute the Layer-3 topology. The factual cost of the mining task is the product of the above factors, which is obviously not affordable in real networks.

On the other hand, mining specification with analysis-based verifiers might mitigate the scalability problem, but sacrifice fidelity: instead of generating the RIBs and FIBs through simulation, the analysis-based verifiers could check the properties and the corresponding failure tolerance levels based on control plane models [1], [3], [6], [23]. Although the one-time analysis takes more time than simulation, this approach only requires $O(S^2)$ or even fewer rounds of verifications (S is the number of subnets), making it much more scalable than the simulation-based approach. However, the models used by analysis-based verifiers abstract away some important features, and therefore cannot faithfully reflect the behaviors of real network devices.

We believe fidelity should be the top concern, because one missing property can cause false negatives in the verifiers and/or fail the critical applications if used in synthesizers. This urges us to use the simulation-based verifier for the highest-level fidelity, and leaves the key question: *can we mine the specification with high scalability and without loss of fidelity?* To this end, we need to conquer the aforementioned two scalability factors. By revisiting the simulation workflow, we have the following observations.

Observation 1: *A limited number of failure scenarios are related to a given property.* Consider two properties with the forwarding paths $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow D$, respectively. It is easy to see that when $l = 1$, we only need to check three failure scenarios: failure of BC for the first,

979-8-3503-0322-3/23/$31.00 ©2023 IEEE

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

failure of BD for the second, and failure of AB for both. That is, other possible failure scenarios, say failure of AD, are irrelevant to *all* candidate properties, and thus can be skipped. We emphasize this principle could eliminate a considerable number of scenarios, because when l is relatively large, say $l = 2$, each property can only relate to a handful number of scenarios. As such, most scenarios can be skipped without simulations.

Observation 2: *Layer-1 modeling dominates the simulation.* To maintain high fidelity, the specification mining must consider physical failure scenarios at Layer-1[1], instead of logical failure scenarios at Layer-3. We observe that computing the Layer-3 topology is surprisingly more costly ($\sim 200\times$ according to our experiments §V-B) than computing routes based on Layer-3 topology. The reason is mostly due to the large number of physical ports, each configured to allow a large number of VLANs. As such, reducing the cost of Layer-1 modeling could save considerable time during one simulation.

With the above observations, we propose *NetMiner*, a tool for mining network specification, to address this problem. *NetMiner* chooses the simulation-based approaches (e.g., Batfish) such that it will not sacrifice any fidelity. Specifically, *NetMiner* makes the following contributions, to reduce the number of scenarios and the cost of the one-time simulation.

Contribution 1: General Scenarios Aggregation. *NetMiner* designs a *General Scenarios Aggregation* method that eliminates the irrelevant failure scenarios. First, *NetMiner* identifies the property-related failure scenarios in a general way. We say a scenario relates to a property if it contains at least one link from the *forwarding path* (traversed by the packets related to the property) *or* the *routing path* (traversed by the routes matched by the packets); otherwise, the failure scenario is irrelevant to the property. Then, *NetMiner* derives both the forwarding path and routing path based solely on the routes returned by the simulator, rather than a control plane model. Finally, *NetMiner* aggregates the failure scenarios shared by different properties, largely eliminating the irrelevant scenarios. Compared to the enumeration approach, the aggregation reduces the number of failure scenarios by 2 to 3 orders of magnitude.

Contribution 2: Fast Topology Mapping. *NetMiner* designs a *Fast Topology Mapping* method, which can rapidly generate the Layer-3 topology from the Layer-1 topology. Initially, we construct a bidirectional map between the Layer-1 topology and the Layer-3 topology, which only needs to be built once. When we need to fail Layer-1 links, we can query the map to identify those Layer-3 links that use the Layer-1 links, and determine whether these Layer-3 links will fail. This avoids the need of rebuilding the Layer-3 topology from scratch, each time when the Layer-1 topology changes. Compared with the baseline approach generating Layer-3 topology from scratch used by Batfish, our method can significantly improve the speed of generating Layer-3 topology by 5 to 6 orders of magnitude.

Based on real configurations from a large cloud service provider, and synthetic configurations from prior works, we show *NetMiner* can mine the specification 3 to 4 orders of magnitude faster than the simulation-based method, and surprisingly, 10 $\times$ faster than the state-of-the-art analysis-based method.

[1]We use Layer-1 to represent Layer-1 and Layer-2, and thus the logic of Layer-1 contains physical links aggregation, VLANs, etc.

II. Motivation

In this section, we first motivate the problem of specification mining (§II-A). Then we discuss why simply leveraging existing methods cannot fulfill our goals (§II-C). Finally, we formally define the problem of specification mining from the configurations (§II-B).

A. Why Mining Specification

Enabling network changes verification. Verifying a configuration has the desired effect only requires analyzing targeted header space, e.g., reachability between two subnets. However, checking for unintended side effects is harder, because changes may impact seemingly unrelated header spaces and hidden failure scenarios [4]. Moreover, compressing redundant configurations needs to verify the consistency of the configurations before and after compression, which usually needs to verify all properties across multiple failure scenarios.

Helping human understanding. Network operators often complain about inheriting an already working legacy network, whose intents are hard to tell [24]. The complexity of routing protocols makes it challenging for operators to understand the network intents from low-level configurations. Automatically generating the specification of a network helps operators to better understand what the network is currently doing, and makes network update easier.

Facilitating intent-based networking. Current network management relies heavily on humans. Consequently, manual operations not only increase the burden of administrators, but also increase the network security risks. Generally, inferred intents can be used as input for any intent-based networking tool, e.g., automatic configuration synthesis [8]–[12]. In addition, automatic configuration mining enables automatic migration, e.g., transparent migration from legacy networks to SDN, other vendor networks or cloud platforms [25].

B. Problem Definition

The following formulates the problem of *network specification mining*. The formulation is inspired by Config2Spec [23], but differs in how failures are modeled.

Failure model. The network we consider has a set of nodes (physical switches or routers) N, and a set of physical links $links$ connecting the ports of two nodes. A small number of nodes or links are allowed to fail, due to hardware or software failures of nodes or ports. We define a *link failure scenario* as a partition of $links$ into ($links_{up}$, $links_{down}$), which consist of links that are up and down, respectively. Node failures can

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

TABLE I
Network properties (*s*, *n*, *w* are routers, *d* is prefix.)

Property	Description
reachability(*s,n,d*)	packets in *d* sent from *s* can reach *n*
waypoint(*s,n,w,d*)	packets in *d* sent from *s* to *n*, passing through *w*
loadbalancing(*s,n,d,m*)	packets in *d* sent from *s* to *n*, along *m* paths

be viewed as failing all the links that are connected to the ports of the nodes. Without extra explanation, the rest of the paper will use *scenario* to denote a link failure scenario. Note that our model considers failures of physical (Layer-1) links, rather than logical (Layer-3) links, as considered by Config2Spec and control plane verifiers like NetDice, etc. This makes the failure model more realistic.

Intent and Property. An intent I is defined as $p : t$, where p is a property, and t is the failure tolerance of p. Table I lists the properties considered in this paper, including reachability, waypoint and load balancing. The failure tolerance t is an integer representing the number of link failures. For example, $Reachability(s, n, d) : 3$ means packets of d sent from source s can reach destination n when there are no more than 3 link failures. We use the notation $mt(p)$ to denote the maximum failure tolerance of a specific property p. Given network configurations C, its specification $Spec^C$ is a collection of intents that are defined as:

$$Spec^C = \{p : mt(p) | p \in P^C\} \tag{1}$$

where P^C is the set of all properties that hold without node or link failures. Note here, P^C can also be limited to a subset of all properties, which operators care about.

We usually consider a small number (say less than 6) of link failures, which is mostly enough for operators (when not accounting for link aggregations). Therefore, we have the following compromised version for specification:

$$Spec^C_L = \{p : min(L, mt(p)) | p \in P^C\} \tag{2}$$

where L is the bound on the maximum number of link failures.

The problem of network specification mining is defined as follows.

PROBLEM 1. *Given network configuration C and a bound L of maximum failures, find the specification* $Spec^C_L$.

C. Limitations of Existing Methods

Except for mining specification from the configurations, one can also mine specification from (1) network states (e.g., FIB or RIB) with data plane verifiers [19], [20], or (2) network behaviors (e.g., network traffic) [24], [26]–[29]. However, these methods may miss intents like "A subnet is always reachable under a single link failure". In this paper, we focus on mining from configurations, as this method can reveal the complete set of properties.

Property-Aggregation Approaches. Most simulation-based verifiers, e.g., Batfish [15], CrystalNet [16], ShapeShifter [17], Plankton [2] and DNA [18] can generate FIBs and RIBs for a certain scenario, and the data plane verifiers [19]–[21] can then rapidly check multiple (all) candidate properties on that scenario. Such a prototype aggregates the properties but has to enumerate all failure scenarios, whose number amounts to $O(C^l_L)$, where l is the failure tolerance level and L is the number of links.

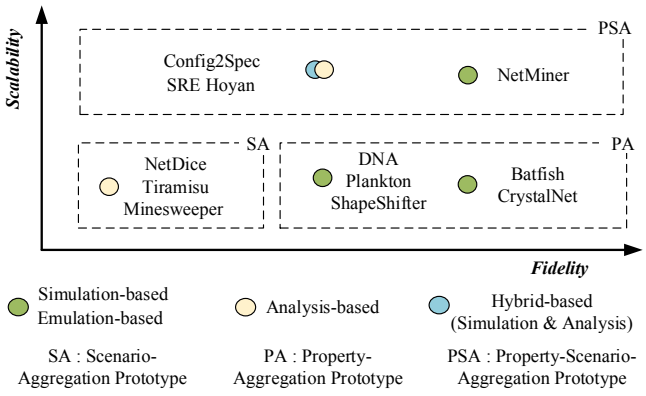


Fig. 1. The tools available for mining specification.

Scenario-Aggregation Approaches. Analysis-based verifiers like Tiramisu [1], NetDice [3] and Minesweeper [6] can check if one certain property holds for all failure scenarios. Building upon these tools avoids the link failure enumeration, but in turn needs to enumerate the properties, which leads to $O(S^2)$ model calculations in an S-subnet network. More importantly, all these works have a lower fidelity than the simulation-based approach. For example, Minesweeper has not modeled features like multiple virtual routing and forwarding (VRFs) and multi-hop BGP neighbor; Tiramisu does not support features like protocol- or port-based packet filters, packet filters impacting route advertisements and route filters matching multiple community tags.

Property-Scenario-Aggregation Approaches. Analysis-based methods like Hoyan [5] and SRE [4] combine header space and failure space to improve the scalability of verification in larger and complex networks. Config2Spec [23] is the closest work to us, which iterates over the simulation-based verifier (Batfish [15]) and analysis-based verifier (Minesweeper [6]) by the predicted cost. These three tools can analyze multiple properties over multiple scenarios. However, such scalability improvement comes from the control plane models used in the analysis-based verifiers. Therefore, they all suffer from fidelity defects. For example, all the above approaches only consider logical failures, i.e., failure of Layer-3 links, and as a result, would over- or under-estimate the failure tolerance of properties. Taking Fig. 2(a) as an example, which is based on Layer-3 topology. Since there are two paths from $r1$ to $d1$, the property $p1$ has a failure tolerance $mt = 1$; However, if considering the Layer-1 topology shown in Fig. 2(b), $p1$ should have a failure tolerance $mt = 0$. The reason is that $r1$ can only reach $d1$ via VLAN 1 through the path $r1 \rightarrow s1 \rightarrow r3$.

To consider physical link failures, we need to have a way to determine which Layer-3 links would be affected once physical links fail. This is not easy since the mapping between

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

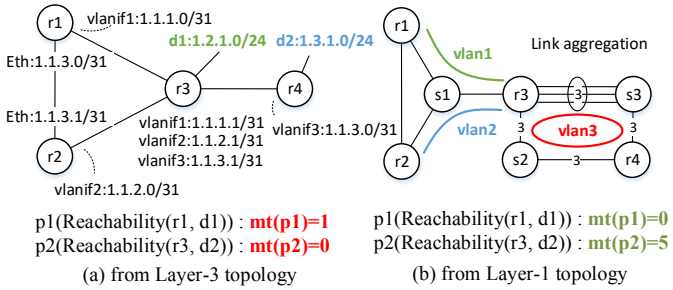


Fig. 2. An example illustrating the differences when mining specifications based on Layer-1 topology and Layer-3 topology. Subnets $d1$ and $d2$ are attached to router $r3$ and router $r4$, respectively. All devices are in the same OSPF domain and the weights of all links are the same.

Layer-1 links and Layer-3 links is complicated. For example, failing Layer-1 link $s1-s3$ affects two Layer-3 links $r1-r3$ and $r2-r3$, but failing Layer-1 link $r3-s3$ does not affect any Layer-3 links.

In contrast to all previous works, *NetMiner* aims to achieve Property-Scenario-Aggregation with pure simulation-based verifiers.

III. Overview

In this section, we first show the workflow of *NetMiner* and reveal the technical challenges (§III-A). Then we explain how to achieve high scalability with *General Scenarios Aggregation* (§III-B) and *Fast Topology Mapping* (§III-C), without loss of fidelity.

A. Workflow of NetMiner

The workflow of *NetMiner* is shown in Fig. 3, which takes as the input the network configuration C, *NetMiner* outputs the formal specification $Spec_L^c$ that consists of property p and the corresponding failure tolerance level mt. *NetMiner* decouples the vendor-dependent behaviors (*driver*) and vendor-independent behaviors (*core*), which ensures its fidelity.

In its core layer, *NetMiner* first calculates the initial property space P^c based on the configuration and physical topology to form a set of *unverified properties (UP)* (①).

NetMiner then verifies *UP* with different failure tolerance levels, starting from an all-link-up Layer-1 scenario, i.e., $l = 0$. Then, *NetMiner* maps the current Layer-1 scenario (*L1s*) into the Layer-3 scenario (*L3s*) (②), which, together with the configurations, would be fed into the simulation-based control plane verifier. The simulation produces a concrete data plane for *L3s* (③), and then the data plane verifier can check whether the whole *UP* is satisfied under *L3s*, i.e., Property-Aggregation (④).

If *UP* does not hold, *NetMiner* would update the tolerance of the property in *UP* with current l and add *UP* into the specification.

Otherwise, *NetMiner* produces the failure scenarios for the next tolerance level. In doing so, *NetMiner* first calculates the *hot links* (⑤), i.e., links that form the forwarding path and routing path, of *L3s* for each property in *UP*. These links form a property-related Layer-3 scenarios, which will be further mapped back into Layer-1 scenarios (⑥). Through

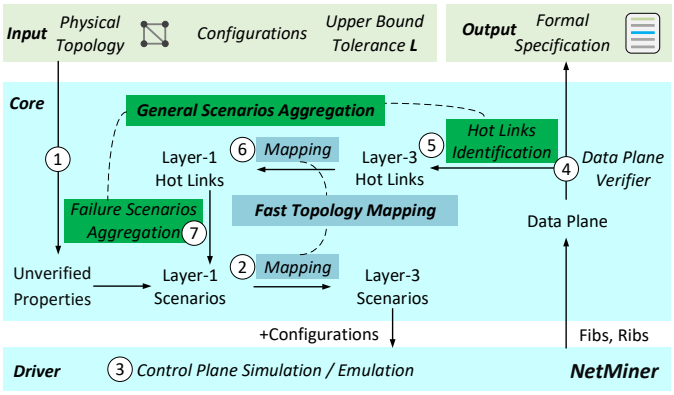


Fig. 3. The workflow of *NetMiner*.

this process, *NetMiner* identifies and aggregates the Layer-1 scenarios shared across the properties in *UP*. Finally, each Layer-1 scenario from the aggregated scenarios becomes the new *L1s*, and the properties related to it form the new *UP* (⑦). The mining process recursively iterates until l reaches L.

We highlight two technical challenges in this workflow. First, when calculating the *hot links*, *NetMiner* must not involve any vendor-dependent control plane behaviors to avoid loss of fidelity, i.e., it should be based purely on the output of the control plane (e.g., RIBs, FIBs). This is challenging considering the complex recursive route resolution process, the existence of static routes, and route reflectors, etc. Second, Layer-1 and Layer-3 mapping is frequently invoked through this process, which, as mentioned in §I, dominates the one-time verification. *NetMiner* designs two key modules, *General Scenarios Aggregation (GSA)* and *Fast Topology Mapping (FTM)* for addressing the above two challenges. We present these two modules with concrete examples in the following sections.

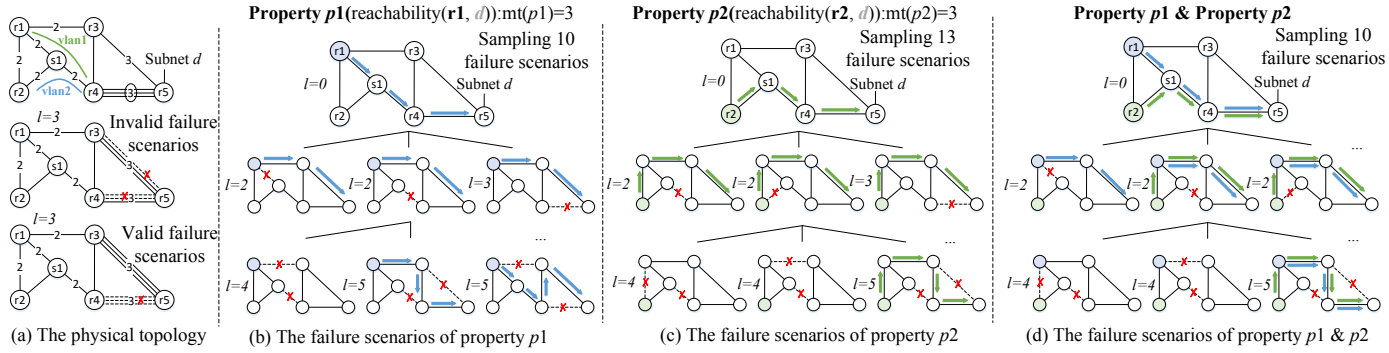
B. General Scenarios Aggregation

Properties like reachability and waypoint cannot be held if there do not exist any physical links from the forwarding path and the routing path. We call links that form these two paths *hot links*. The failure scenarios that do not contain any hot link are considered irrelevant to the property.

Hot links identification. To identify all *hot links*, the straightforward way is to model the control plane behavior, including importing or redistributing routes, filtering routes, selecting the best routes, etc., to derive the related links of a property [3]. However, control plane behaviors are vendor-dependent, making it hard to model them correctly. For example, different vendors have different processes for selecting the best routes; most vendors need a route to exist in the main RIB so that it can be imported to BGP, while some vendors do not require the existence of routes. Therefore, identifying the *hot links* based on a control plane model can impact fidelity.

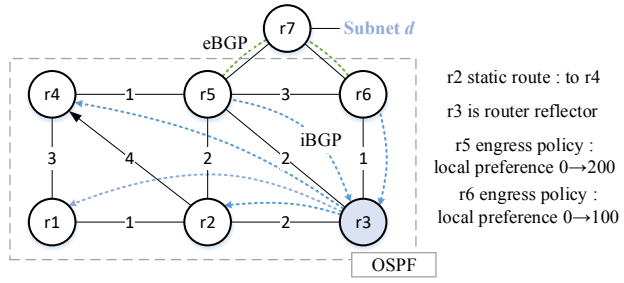
We observe that the vendor-dependent behaviors, i.e., route propagation and best route selection, are already handled after the simulators compute the routes, i.e., routing table. This reveals the chance to trace forwarding paths and routing paths, purely from the rules of the routing table, e.g., the routing

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

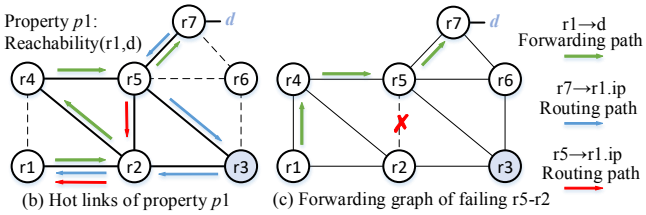


(a) The physical topology (b) The failure scenarios of property $p1$ (c) The failure scenarios of property $p2$ (d) The failure scenarios of property $p1$ & $p2$

Fig. 4. An example of failure scenarios aggregation. The network contains five routers and one switch. The numbers on the links represent the number of physical links aggregated. Device $r1$ and $r4$ are reachable via VLAN1 and device $r2$ and $r4$ are reachable via VLAN2.



(a) An example of a network (The number in edges represent ospf cost).



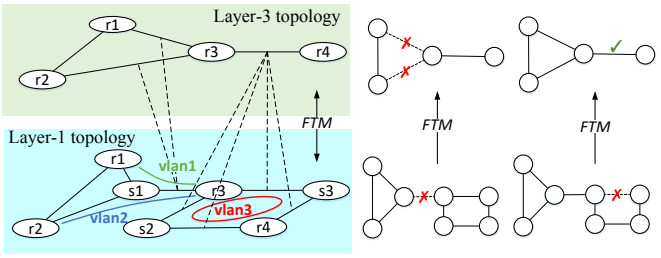
(b) Hot links of property $p1$ (c) Forwarding graph of failing r5-r2

Fig. 5. The network contains seven network devices and a directly connected route to d exists on device $r7$. $r5$ and $r6$ have an eBGP neighbor relationship with $r7$, respectively, while $r3$ is a route reflector. The remaining six devices, except $r7$, are configured with the OSPF protocol.

last hop and the forwarding next hop. One critical point through this process is that we need to model the routing recursion semantics such as iBGP's dependency on OSPF for the completeness of *hot links*. We detail this algorithm in §IV-A and prove it preserves the fidelity.

Failure scenarios aggregation. After identifying the *hot links* for each property, *NetMiner* generates their corresponding failure scenarios and simulates the control plane under each failure scenario. However, directly simulating all failure scenarios for each separate property still results in a large number of simulations. Here, we observe that many failure scenarios are shared by multiple properties, and thus *NetMiner* first aggregates them and simulates each of these failure scenarios once, significantly reducing the total number of simulations. Taking Fig. 4(a) as example, and consider two reachability properties $p1$ ($r1 \rightarrow d$) and $p2$ ($r2 \rightarrow d$), with a bound $L = 3$ on failure tolerance. Fig. 4(b) and (c) show the failure scenarios that need to be analyzed for $p1$ and $p2$, respectively. Without aggregation, we need to analyze 23 failure scenarios. However, 7 out of these failure scenarios are shared by both properties, and we can analyze them together without a single simulation. As a result, the total number of simulations reduces from 23 to 16.

In real cases, the reduction can be quite significant: Fig. 10 shows that failure scenarios aggregation can reduce the total number of simulations by 1-2 orders of magnitude (see §IV-A for details).



(a) Birdirectional topology mapping (b) Example of using topology mapping

Fig. 6. An example of Fast Topology Mapping.

A walk-through example. We show an example in Fig. 5(a) to illustrate how we identify the *hot links*, and hence reduce the failure scenarios. We consider the reachability property $p1$ from device $r1$ to subnet d. (1) For forwarding paths, we start from device $r1$, and find the next hop to d is $r5$ in BGP routing table. Because this is a recursive route, we then find the next hop on $r1$ to ip_{r5} is $r2$ in OSPF routing table and the next hop of $r2$ to ip_{r5} is $r4$ in static routing table. With the same steps, we finally get the forwarding paths as $r1-r2-r4-r5-r7$. (2) For routing paths, we also start from device $r1$, we find the routing last hop of $r1$ to d is $r3$ (the simulator could provide this information) in BGP routing table. Then we find the forwarding path from $r3$ to ip_{r1} is $r3-r2-r1$ in OSPF routing table (the step is similar to (1)). We next know the last hop of $r3$ to d is $r5$ in BGP routing table. We finally get the routing path as $r1-r2-r3-r5-r7$. We calculate the *hot links* in Fig. 5(b). In this example, the simulator helps us to model a modified routing announcement, i.e., the simulator tells the forwarding next hop of $r1$ to d is $r5$ instead of $r6$ (avoid modeling vendor-dependent behavior). But we still need to consider the vendor-independent routing recursion. As in Fig. 5(c), if we don't consider routing recursion, the link $r2-r5$ would be ignored and actually affect the forwarding behavior.

C. Fast Topology Mapping

NetMiner avoids the re-computation of Layer-3 topology using *Fast Topology Mapping* (*FTM*). It is based on two

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

TABLE II
Notations used for hot link identification.

Notations	Meaning
$Path(s,d)$	forwarding path from source s to prefix d.
$Info(s,d)$	the best route for d on s, including next-hop and attributes.
$H(s,d)$	set of hot links from source s to prefix d.
$F(s,d)$	set of links whose failures will change $Path(s,d)$.
$R(s,d)$	set of links whose failures will change $Info(s,d)$.

observations: (1) Each layer-3 link is associated with a few number of Layer-1 links, and vice versa. (2) The semantics of Layer-2, including VLAN, STP, etc., are relatively simple (compared with Layer-3 route computation) Therefore, we can construct a bidirectional map between Layer-1 topology and Layer-3 topology, to avoid re-computing Layer-3 topology from scratch each time a Layer-1 link is failed.

Initially, *NetMiner* constructs the Layer-3 topology based on the Layer-1 topology, meanwhile computing two maps: one from a Layer-1 failed link to a Layer-3 failed link, and one from a Layer-3 hot link to a Layer-1 hot link. Then, each time *NetMiner* needs to fail a Layer-1 link, it can quickly generate the Layer-3 topology based on the first map. When *NetMiner* has identified some Layer-3 *hot links*, it can use the second map to obtain the corresponding Layer-1 *hot links*.

In real cases, the mapping can significantly accelerate the simulation under failure scenarios, e.g., 5-6 orders of magnitude faster as shown in Table IV.

A walk-through example. We maintain bidirectional mapping when constructing the Layer-3 topology with all Layer-1 links up. This process only needs to be done once. As shown in Fig. 6(b), suppose one needs to fail a link $s1-r3$ at Layer-1, after looking up in the mapping, both $r1-r3$ and $r2-r3$ will be affected at Layer-3. Then we find that the paths $r1-s1-r3$ and $r2-s1-r3$ of these two links at the Layer-1 topology all are blocked, so both links failed. For another example, when $s3-r3$ in Layer-1 is failed, we first determine that $r3-r4$ in Layer-3 will be affected, and then find that there is another path, i.e., $r3-s2-r4$ in Layer-1. Then, no Layer-3 links are affected.

IV. Design Details

This section presents the details of *General Scenarios Aggregation* (§IV-A), and *Fast Topology Mapping* (§IV-B), followed by several optimizations (§IV-C).

A. General Scenarios Aggregation

The computation of failure scenarios relies on identifying *hot links* whose failure will change the forwarding or routing paths. In the following, we first define *hot links*, and then show how to compute relevant failure scenarios by identifying hot links, and finally show how to aggregate these scenarios to improve scalability.

Definition 1. *Given a property $p(s,d)$ and failure scenario $f(links_{up}, links_{down})$, a link $link \in links_{up}$ is "hot" with respect to f and p, iff failing $link$ will change the forwarding behavior (paths) of the packets from s to d.*

Algorithm 1: Hot Link Identification

input : s: the source device, d: the destination prefix, $Rib^{\mathcal{P}}$: the RIB for protocol $\mathcal{P}$, Rib: the Main RIB.
output: $\mathcal{H}$: the set of hot links.

1 **Function** $H(s,d)$**:**
2 **return** $R(s,d) \cup F(s,d)$
3 **Function** $F(s,d)$**:**
4 $\mathcal{H} \leftarrow \{\}$
5 **if** $Connected(s,d)$ **then**
6 **return** $(s, Node(d))$ ▷ s have a direct route to d
7 $\mathcal{P} \leftarrow Rib_s(d).Type$ ▷ the protocol of route to d
8 $D \leftarrow \{r.NextHop | r \in Rib_s^{\mathcal{P}}(d)\}$ ▷ ECMP
9 **foreach** $d' \in D$ **do**
10 $\mathcal{H} \leftarrow \mathcal{H} \cup H(s,d')$
11 **if** $Node(d') \neq Node(d)$ **then**
12 $\mathcal{H} \leftarrow \mathcal{H} \cup H(Node(d'),d)$ ▷ not on same device
13 **return** $\mathcal{H}$
14 **Function** $R(s,d)$**:**
15 $\mathcal{H} \leftarrow \{\}$
16 **if** $Connected(s,d)$ **then**
17 **return** $(s, Node(d))$
18 $\mathcal{P} \leftarrow Rib_s(d).DynType$ ▷ dynamic routing type
19 $D \leftarrow \{r.LastHop | r \in Rib_s^{\mathcal{P}}(d)\}$ ▷ ECMP
20 **foreach** $d' \in D$ **do**
21 $\mathcal{H} \leftarrow \mathcal{H} \cup H(Node(d'), s.peer\_ip)$
22 **if** $Node(d') \neq Node(d)$ **then**
23 $\mathcal{H} \leftarrow \mathcal{H} \cup R(Node(d'),d)$
24 **return** $\mathcal{H}$

Identifying hot links. For each property, *NetMiner* identifies all the *hot links* solely based on the RIBs computed by simulators (e.g., Batfish). This makes the hot link identification fully agnostic of vendor-specific protocol implementations, which has already been accounted for when simulators compute the routes, and therefore does not hurt fidelity. In contrast, *NetDice* [3] designs a customized algorithm to model the route computation process, and the *hot links* may not be correct due to vendor-dependent behaviors.

NetMiner recursively resolves the forwarding path of the packets, and the routes that are used during the forwarding. At the same time, *NetMiner* resolves the path that those routes are propagated. Then, *NetMiner* classifies all the links on either the forwarding path or routing path as "hot", and computes relevant failure scenarios by failing one of those *hot links* each time.

Alg. 1 shows this process, where the symbols are defined in Table II. Given a source node s, a destination prefix d, and a set of routing tables, the algorithm outputs a set of *hot links* $\mathcal{H}$. First, the *hot links* appear on either the forwarding path or the routing path (Line 1-2). For the forwarding paths, if s has a directly connected route to the prefix d, then the algorithm returns the link from s to the next-hop device (Line 5-6); Otherwise, the algorithm resolves the next-hop IP addresses by looking up the RIBs of the protocol (Line 7-8). For each next-hop IP address d', the algorithm recursively computes the

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

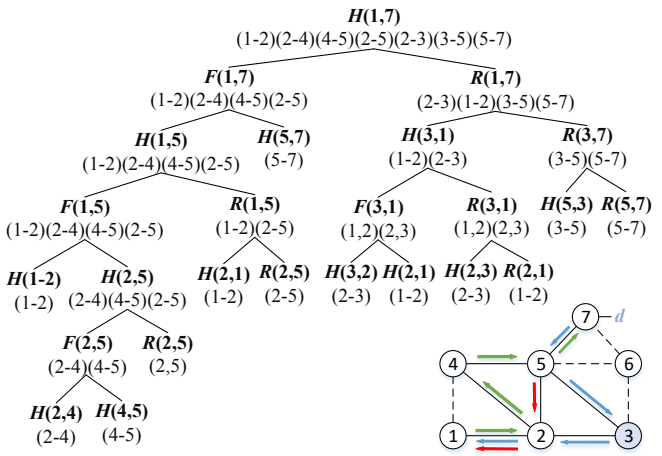


Fig. 7. The process of Alg. 1 running on the network of Fig. 5. We simplified the function parameters in the graph. We omit the leaf nodes in the graph, which are the functions F and R.

forwarding path from s to d' (Line 9), and also the forwarding path from the device directly connected to d' to the destination prefix d (Line 11-12). The forwarding path would be the union of the above two forwarding paths (Line 13). The process of computing routing paths is very similar (Line 14-24). The difference is that: (1) only dynamic routes (OSPF, BGP, etc.) are considered since static routes do not depend on any links (Line 18). (2) instead of the next-hop IP address, the algorithm resolves the IP address that the last-hop router $Node(d')$ uses to send the route to the current node s (Line 19); (3) the reachability of the route recursively depends on the packet reachability from $Node(d')$ to s (Line 21).

Fig. 7 shows the tree when using Alg. 1 to compute *hot links* for the previous example (Fig. 5). The algorithm starts from root node $H()$ and recursively calls functions $F()$ and $R()$, which may recursively call functions $H()$ and $R()$. At each node, the set of *hot links* is shown below the function.

THEOREM 1. *For a p(s, d) and a failure scenario f. The set of links $\mathcal{H}$ returned by Alg. 1 contains all hot links for p(s, d) on f.*

Proof: Due to the lack of space, we only show the proof sketch here. First, we show the theorem holds when there is only OSPF: the theorem holds when the subnet d is one-hop away from s; and if the theorem holds for k-hops away, then it also holds for $(k+1)$-hops away. Then, we include the cases for static routes, OSPF, and BGP. The proof is similar except that the BGP route may be indirect (depending on OSPF or static routes). ■

Aggregating failure scenarios. After identifying the *hot links*, *NetMiner* generates failure scenarios for each property, and aggregates the common failure scenarios. To realize the aggregation, *NetMiner* maintains a set UP of *unverified properties*, and a set F_l of *failure scenarios* for each tolerance level l. Each $f \in F_l$ is registered to a set of properties $prop(f)$, meaning that *NetMiner* needs to simulate f to determine the failure tolerance levels for these properties. Let $\mathcal{L}1$ be the set of all Layer-1 links. Initially, $F_0 = f(\mathcal{L}1, \emptyset)$, and $prop(f) = UP$ consists of all properties that hold under no failures. *NetMiner* simulates the control plane under failures in F_l starting from $l = 0$, and retrieves the FIBs. Then, for each property $p \in prop(f)$, *NetMiner* checks if p holds based on the FIBs. If p holds, then *NetMiner* identifies the Layer-3 *hot links* with Alg. 1, and maps the Layer-3 *hot links* to Layer-1 *hot links* with *FTM*. Let $link^1$ be such a hot link, which aggregates num physical links. Then, *NetMiner* moves p from $prop(f)$ to a new $prop(f')$, where $f' \in F_{l+num}$. Otherwise, if p does not hold, then, *NetMiner* moves p from $prop(f)$ to the set of verified properties VP, and sets the tolerance level of p to l. The above process continues when $F_l = \emptyset$ for each $l \leq L$.

Simulating the control plane under failures. For each failure scenario $f^1(links^1_{up}, links^1_{down})$ (a partition of layer-1 links as defined in §II-B), We derive the corresponding Layer-3 failed links $links^3_{down} = M^{1\to3}(links^1_{down})$, where $M^{1\to3}$ is a map from Layer-1 links to Layer-3 links (*FTM* as defined in the following subsection.) Then, we feed the Layer-3 failure scenario $f^3(links^3_{up}, links^3_{down})$ to off-the-shelf control plane simulator or emulator, and retrieve the forwarding tables (FIBs) and routing tables (RIBs).

B. Fast Topology Mapping

NetMiner needs to compute what Layer-3 links will fail when failing a Layer-1 link, for the control plane simulator (e.g., Batfish) (Task 1). This is time-consuming if *NetMiner* directly uses Batfish to re-compute a new Layer-3 topology. In addition, after *NetMiner* uses Alg. 1 to identify the Layer-3 *hot links*, it needs to compute the Layer-1 *hot links* (Task 2). To efficiently support the above two tasks without re-computation, *NetMiner* constructs bidirectional maps between Layer-1 links and Layer-3 links.

Initializing the Layer-3 topology. Let $\mathcal{L}1$ be the set of all Layer-1 links. [2] Then, *NetMiner* constructs the set of all Layer-3 links $\mathcal{L}3$, such that $(s, d) \in \mathcal{L}3$ iff the following conditions are satisfied: (1) an interface of s and another interface of d are in the same subnet; (2) these two interfaces are reachable through some VLAN; (3) there is at least a physical path (a sequence of Layer-1 links) between these two interfaces. We initialize the Layer-3 topology in the following two steps:

Step 1: Layer-2 topology construction. Given $\mathcal{L}1$, *NetMiner* obtains the virtual links between Layer-2 port and Layer-1 port from the network configurations, and then derives the links between Layer-2 ports. As shown in Fig. 8, R2-Eth2 and S1-Eth4 are two connected Layer-2 ports.

Step 2: Layer-3 topology construction. NetMiner creates virtual links between each Layer-3 port with all Layer-2 ports on the same device. Then, *NetMiner* constructs the Layer-3 links $\mathcal{L}3$ by simulating the forwarding of Layer-2 frames according to VLAN numbers. As shown in Fig. 8, *NetMiner* constructs a Layer-3 link (R2-Vlanif2, R3-Vlanif2) since their

[2]One viable method of obtaining $\mathcal{L}1$ is to use the SNMP protocol to read the Physical Topology MIB of each device [30].

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

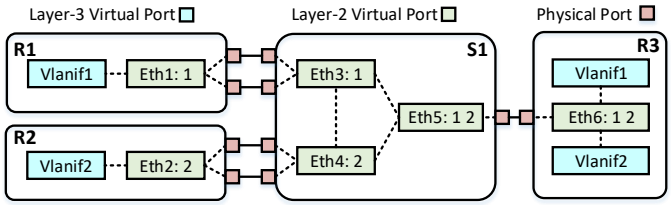


Fig. 8. An illustration of Layer-3 topology construction for Fig. 6. Here, dashed and solid links represent the virtual links we create and the real links, respectively.

frames can traverse a path R2-Eth2 → S1-Eth4 → S1-Eth5 → R3-Eth6.

During the construction of Layer-3 topology, *NetMiner* also computes two maps. For each $l^3 \in \mathcal{L}3$, let $Paths(l^3)$ be the set of all physical paths for l^3. For example, in Fig. 6, $Paths((r3, r3)) = \{r3 - s3 - r4, r3 - s2 - r4\}$. For each $l^1 \in \mathcal{L}1$, let $Links(l^1)$ be the set of Layer-3 links that "use" l^1. Formally, $Links(l^1)$ is defined as:

$$Links(l^1) = \{l^3 | \exists path \in Path(l^3), l^1 \in path\} \quad (3)$$

NetMiner needs to re-initialize the Layer-3 topology if the physical topology changes, which is quite infrequent. As a result, we believe a minute-level running time for such a task is affordable in most cases.

Defining the bidirectional maps. After the initial construction of Layer-3 topology, *NetMiner* can realize the bidirectional map between Layer-1 links and Layer-3 links:

- $M^{1\to3} : \mathcal{L}1 \to 2^{\mathcal{L}3}$, which maps Layer-1 links L^1 to a set S of Layer-3 links, satisfying that if links in L^1 fail, then Layer-3 links in S will fail.
- $M^{3\to1} : \mathcal{L}3 \to 2^{\mathcal{L}1}$, which maps a Layer-3 link l^3 to a set of S Layer-1 links, satisfying that failing any other links not in S will not affect l^3.

Task 1: From Layer-1 failed links to Layer-3 failed links. Each time *NetMiner* needs to simulate the control plane after failing a Layer-1 link L^1, it uses Eq. (4) to compute the Layer-3 failed links $M^{1\to3}(L^1)$, without re-computing the Layer-3 topology using simulators (e.g., Batfish).

$$M^{1\to3}(L^1) = \{l^3 \in Links(l^1) | l^1 \in L^1, \nexists path \in Path(l^3), path \wedge L^1 = \varnothing\} \quad (4)$$

The time complexity for computing Eq. (4) is $O(KNM)$, where K is the size of L^1, N is the average size of $Links(l^1)$, and M is the average number of links in $Path(l^3)$.

Task 2: From Layer-3 hot links to Layer-1 hot links. After *NetMiner* identifies the Layer-3 *hot links*, for each hot link l^3, it uses Eq. (5) to identify the corresponding Layer-1 *hot links* $M^{3\to1}(l^3)$.

$$M^{3\to1}(l^3) = \{l^1 \in \mathcal{L}1 | \exists path \in Paths(l^3), l^1 \in path\} \quad (5)$$

The time complexity for computing Eq. (5) is $O(M)$.

This approach may over-estimate the *hot links* when there are multiple reachable paths on Layer-1, and the network will use Spanning Tree Protocol (STP) to select one of the paths. Currently, we find no performance degradation due to the over-estimation, and therefore have not considered the simulation of STP.

C. Optimization - Property trimming

We initially use the P^C (See §II-B) as *unverified properties (UP)* that *GSA* needs to consider. We reduce the number of properties that *GSA* needs to consider with two trimming methods, and thus reduce the number of failure scenarios in *NetMiner*.

Trimming based on enumeration analysis. When the number of failed links l is relatively small, the enumerated failure scenarios are basically the same as the failure scenarios calculated by *GSA*. At this time, we can set the tolerance threshold L_t and enumerate the failure scenarios when l is below the threshold, and select the failure scenarios by *GSA* when l is above the threshold. For a network without link aggregation, we can simply set $L_t = 1$; Otherwise, we can set L_t to the minimum number of aggregated links between device pairs. When we finish traversing all failure scenarios with enumeration, the tolerance of all properties is determined in the range $0 - L_t$. Then, *NetMiner* takes the tolerance value of property equal to L_t as *unverified properties*, because these properties might have large failure tolerance.

Trimming based on topology condition. Then, we filter properties that do not meet topology conditions in *unverified properties*. If the tolerance value of a property is l, then its minimum cut must be $l+1$. Therefore, we select the minimum cut of property greater than $L_t + 1$ in *unverified properties* to be verified. And there are various efficient ways to compute the minimum cut such as $k + 1$ connected components.

V. Evaluation

In this section, we evaluate *NetMiner* on multiple topologies to address the following questions:

- How does *NetMiner* scale to the actual topologies compared to state-of-the-art? Experiments show that *NetMiner* is 2-5 orders of magnitude faster than the Property-Aggregation approach with simulation-based verifiers, and also an average of 10× faster than the analysis-based approach.
- How do *General Scenarios Aggregation* (*GSA*) and *fast topology mapping* (*FTM*) contribute to *NetMiner*? Experiments show that *GSA* reduces the number of failure scenarios by 2-3 orders of magnitude compared to Property-Aggregation approach. *FTM* can improve the speed of generating Layer-3 topology by 5-6 orders of magnitude compared to the baseline approach generating Layer-3 topology.
- Can *NetMiner* support various configuration features without fidelity loss? Experiments on real configurations show that *NetMiner* avoids some incorrect results returned by Config2Spec.

Implementation. We implement *NetMiner* with 8*k* lines of C++ code and an extra 2*k* lines to re-implement Delta-net

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

TABLE III
Time for *NetMiner* to mine specifications from DC1 and DC2 networks. We set $l = 3$ for the datasets without link aggregation (PHYS) and $l = 12$ for the dataset with PHYS.

Datasets	W/O ALL (s)	W/ ACL (s)	W/ ACL-PHYS (s)
DC1	18901	23004	16341
DC2	733	990	1083

[20]. We use Batfish[3] [15] to generate the data plane (*RIBs and FIBs*) and Delta-net to model the data plane. We extend Delta-net to support load-balancing, and use Delta-net [20] for single-domain incremental updates to build separate models for source and destination IPs. However, our framework can be easily extended to multiple domains, such as APKeep [19].

Approaches for comparison. We implement a Property-Aggregation (PA) approach, i.e., enumerating failure scenarios, with Batfish as the control plane simulator, and Delta-net as the data plane verifier. We implement a Scenario-Aggregation (SA) approach, i.e., enumerating candidate properties, with the open-source code of Tiramisu [31]. We use Config2Spec [32] as the state-of-the-art Property-Scenario-Aggregation (PSA) approach.

Datasets. We use the following three real datasets and three synthesized datasets.

(1) Real configurations of two data center networks (DC1 and DC2) from a large public cloud provider. The configurations include OSPF, BGP, VRF, VLAN, ACL and link aggregation. DC1 (DC2) has 178 (373) routers, 5314 (8673) physical links, $\sim 0.5k$ ($0.7k$) prefixes, $\sim 3k$ (0) ACL rules, single (multiple) VRF, and $\sim 200k$ ($374k$) lines of configurations. Based on each dataset, we construct two additional datasets for comparison with other tools. (1) W/O ALL, by removing ACL, VLAN, and link aggregation from the configurations (2) W/ACL, by removing VLAN and link aggregation from the configurations. Moreover, we note that the initial configurations with ACL, VLAN, link aggregation as W/ACL-PHYS. For W/ACL-PHYS, we set l=12 and others set l=3, which is due to the existence of link aggregation (many device pairs aggregate 4 physical links).

(2) Real configurations of Internet2 network running ISIS, from Config2Spec [23]. The network consists of 10 routers and 18 links.

(3) Synthesized configurations for three WAN networks running BGP or OSPF, from Config2Spec [23]. The BGP (OSPF) datasets consist of small, medium and large topologies, which are 33 (48), 70 (85), and 158 (189) routers (links), respectively.

All experiments run on a Linux server with two 12-core Intel Xeon CPUs @ 2.3GHz and 256G memory.

A. Scalability

We use *NetMiner* and the other three tools to mine specifications from the six datasets. The properties we consider include reachability, waypoint, and load balancing. Since Tiramisu can only verify reachability, we only mine reachability specification for Tiramisu. The bound on maximum link failures is set to $l = 3$. For the DC1 and DC2, we use the dataset W/O ALL, since the other tools cannot correctly process them. We don't run Tiramisu on DC1 and DC2 because of configuration parsing issues.

[3]The version of Batfish involved in all experiments is 0.36.0.

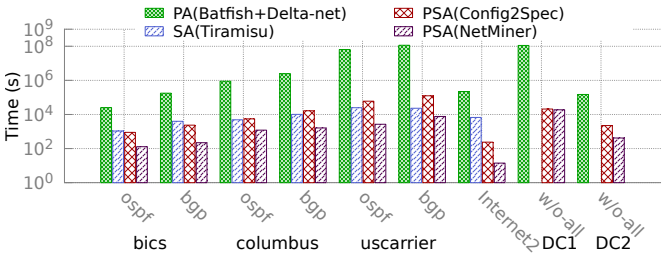


Fig. 9. Time to mine specifications for reachability, waypoint, and load balancing. Here, since PA cannot run to complete within a day, we report the estimated time based on the average time for a single failure scenario and the total number of failure scenarios.

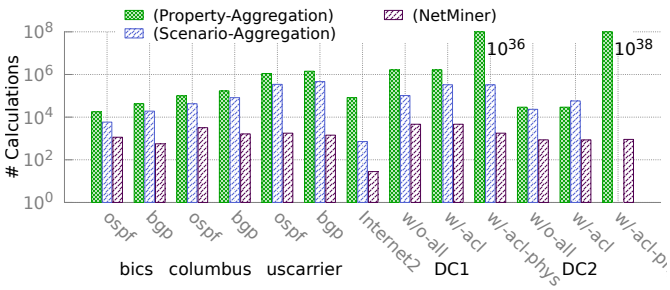


Fig. 10. The number of simulations (for PA and *NetMiner*) or verifications (for SA) of different methods.

Fig. 9 reports the running time. *NetMiner* is 2 to 5 orders of magnitude faster than the Property-Aggregation approach (Batfish+Delta-net), which cannot finish within 2 days, 6 to 16× faster than the Scenario-Aggregation approach (Tiramisu), and 5 to 10× faster than Property-Scenario-Aggregation approach (Config2Spec).

Note that this experiment is not to emphasize the improved performance of *NetMiner*. Instead, our focus here is to show that *NetMiner* does not trade off the scalability for the fidelity, although we use the pure simulation-based verifier.

Mining specifications on the real datasets. Only *NetMiner* can support the real configurations for DC1 and DC2. As shown in Table III, the execution time does not change significantly. For DC1, the time for W/ACL PHYS becomes even faster. The reason is the connections between some devices aggregate 8 or 16 links, making a large number of failure scenarios invalid.

B. Microbenchmark

We show how *GSA* reduces the number of simulations, and how *FTM* reduces the time of one-shot simulation.

General Scenarios Aggregation. We compare the computation cost of different methods, in terms of the number of failure scenarios to simulate (for PA and *NetMiner*), and the number of properties to check (for SA). As shown in

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
Time for computing Layer-3 topology and simulating the control plane. Here, l is the number of failed Layer-1 links.

Datasets	Layer-3 Topology Computation Time				Simulation Time
	Batfish	NetMiner			Batfish
	l=1,2,3	l=1	l=2	l=3	-
DC1	7.8s	9us	11us	15us	3.1s
DC2	192.7s	676us	1076us	1423us	0.54s

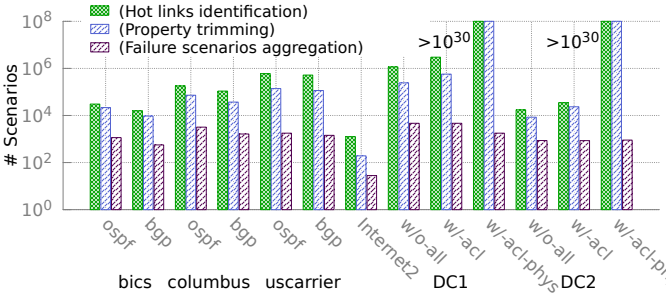


Fig. 11. Comparison of the number of failure scenarios to simulate, after applying hot links identification, property trimming, and failure scenarios aggregation.

Fig. 10, *NetMiner* reduces the number of simulations by 2 to 3 orders of magnitude, compared with PA. The performance gap enlarges when modeling the physical topology, such as W/ACL-PHYS, because when considering link aggregation, we can reduce a large number of invalid failure scenarios. The number of simulations of *NetMiner* is 1 to 2 orders of magnitude less than the number of properties checked by SA. The effectiveness of *GSA* is further shown in Fig. 11. After identifying all *hot links*, the property trimming reduces the number of *unverified properties* by 3 to 6×, while failure scenarios aggregation further reduces the number of scenarios by 1 to 2 orders of magnitude.

In addition, property trimming is only effective with low tolerance level, because for high tolerance property, both trimming method inside property trimming fail at this point.

Fast Topology Mapping. We next show the effectiveness of fast topology mapping (*FTM*). Specifically, we compare the running time for *FTM* and Batfish to generate Layer-3 topology when Layer-1 links are failed. As we can see in Table IV, *FTM* is 5 to 6 orders of magnitude faster than Batfish.

C. Fidelity

We use the two real data center configurations to evaluate the fidelity of *NetMiner*. Specifically, we demonstrate *NetMiner* can avoid incorrect results due to missing features like ACLs, etc., or not modeling physical link failures.

Rich protocol features. *NetMiner* offloads the routing model to the simulation-based control plane verifiers, so that it can easily support rich protocol features as the mature verifiers do, e.g., Batfish. To confirm this, we run Config2Spec and *NetMiner* on DC1 without ACL rules, both of which return a specification with 127079 reachability properties. After adding ACLs into the configurations, *NetMiner* returns 126848 reachability properties, because ACL rules block some traffic,

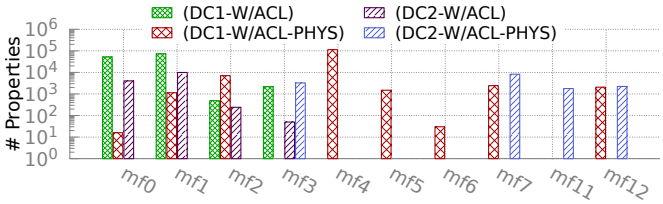


Fig. 12. Distribution of the number of properties for different tolerances between considering physical link aggregation (W/ACL-PHYS) and not considering physical link aggregation (W/ACL).

while Config2Spec still returns the same result. We manually confirmed that the reduction in reachability properties is indeed due to the existence of ACLs.

Real failure model. As shown in Fig. 12, most properties have a high tolerance value when considering physical links rather than logical (Layer-3) links. That is, using existing control plane verifiers [1], [3]–[6], one may under-estimate the failure tolerance of reachability properties. Note that the property with $mt = 3$ in W/ACL may have a higher tolerance value, because we set the upper tolerance limit, so we only mine the tolerance value to $mt = 3$.

VI. Discussion

Supporting more properties. *NetMiner* is extensible to mine any property that can be inferred by the best routes from RIBs. The policies include bounds on path length, multi-path consistency, etc [1], [3]. However, *NetMiner* does not currently support path preference [1], because *NetMiner* does not model non-best routes. At the same time, *NetMiner* cannot mine isolation policy because there is no forwarding path between isolated peer ends.

Supporting virtualized networks. For physical networks, *NetMiner* uses *FTM* to construct the relationship between the Layer-1 topology and the Layer-3 topology, while using *GSA* to derive the reachability between end-pairs on the Layer-3 topology. In the case of virtualized networks, such as VXLANs, additional layers need to be considered. Taking VXLAN for example, we also need to consider the virtual Layer-2 reachability between both ends of the VXLAN tunnel; and virtual Layer-3 reachability between different virtual private networks (VPNs). Extending *NetMiner* to support more layers is left as one of our future works.

VII. Conclusion

We propose *NetMiner*, a network specification mining tool with high scalability and high fidelity. *NetMiner* is built upon pure simulation-based verifiers, so as to abstract away all the vendor-specific models, ensuring its high fidelity. To the end of scalability, *NetMiner* designs (1) a *General Scenario Aggregation* to analyze the property-related scenarios only, and (2) a *Fast Topology Mapping* method that incrementally transforms Layer-1 topology to Layer-3 topology. We evaluate *NetMiner* on real topologies and compare it with the state-of-the-art. Results show that *NetMiner* can scale to large networks while supporting rich protocols and real failure models.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast multilayer network verification," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 201–219.

[2] S. Prabhu, K. Y. Chou, A. Kheradmand, B. Godfrey, and M. Caesar, "Plankton: Scalable network configuration verification through model checking," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 953–967.

[3] S. Steffen, T. Gehr, P. Tsankov, L. Vanbever, and M. Vechev, "Probabilistic verification of network configurations," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 750–764.

[4] P. Zhang, D. Wang, and A. Gember-Jacobson, "Symbolic router execution," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 336–349.

[5] F. Ye, D. Yu, E. Zhai, H. H. Liu, B. Tian, Q. Ye, C. Wang, X. Wu, T. Guo, C. Jin *et al.*, "Accuracy, scalability, coverage: A practical configuration verifier on a global wan," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 599–614.

[6] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 155–168.

[7] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 300–313.

[8] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 328–341.

[9] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "Network-wide configuration synthesis," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 261–281.

[10] ——, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 579–594.

[11] K. Subramanian, L. D'Antoni, and A. Akella, "Synthesis of fault-tolerant distributed router configurations," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–26, 2018.

[12] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock, "Scalable verification of border gateway protocol configurations with an smt solver," in *Proceedings of the 2016 acm sigplan international conference on object-oriented programming, systems, languages, and applications*, 2016, pp. 765–780.

[13] K. Subramanian, L. D'Antoni, and A. Akella, "Synthesis of fault-tolerant distributed router configurations," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–26, 2018.

[14] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Aed: Incrementally synthesizing policy-compliant and manageable configurations," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 482–495.

[15] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 469–483.

[16] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "Crystalnet: Faithfully emulating large production networks," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 599–613.

[17] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Abstract interpretation of distributed network control planes," *Proceedings of the ACM on Programming Languages*, vol. 4, no. POPL, pp. 1–27, 2019.

[18] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *USENIX NSDI*, 2022.

[19] P. Zhang, X. Liu, H. Yang, N. Kang, Z. Gu, and H. Li, "Apkeep: Realtime verification for real networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 241–255.

[20] A. Horn, A. Kheradmand, and M. Prasad, "Delta-net: Real-time network verification using atoms," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 735–749.

[21] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 887–900, 2015.

[22] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 49–54.

[23] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Config2spec: Mining network specifications from network configurations," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 969–984.

[24] A. Kheradmand, "Automatic inference of high-level network intents by mining forwarding patterns," in *Proceedings of the Symposium on SDN Research*, 2020, pp. 27–33.

[25] L. HUANG and L. LU, "Segmentation of ischemic stroke lesion based on long-distance dependency encoding and deep residual u-net," *Journal of Computer Applications*, vol. 41, no. 6, p. 1820, 2021.

[26] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "Netkat: Semantic foundations for networks," *Acm sigplan notices*, vol. 49, no. 1, pp. 113–126, 2014.

[27] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 328–341.

[28] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A language for provisioning network resources," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 213–226.

[29] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 29–42, 2015.

[30] A. Bierman and K. Jones, "Rfc2922: Physical topology mib," 2000.

[31] A. Abhashkumar, A. Gember-Jacobson, and Akella, "Tiramisu source code," 2020, https://github.com/anubhavnidhi/batfish/tree/tiramisu.

[32] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Config2spec source code," 2020, https://github.com/nsg-ethz/config2spec.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:19:43 UTC from IEEE Xplore. Restrictions apply.