Network Specification Mining With High Fidelity, Scalability, and Readability

Ning Kang, Peng Zhang, *Member, IEEE*, Hao Li, *Member, IEEE*, Sisi Wen, Chaoyang Ji, and Yongqiang Yang

***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 pre-condition for network verification and synthesis tools to work. Existing tools for specification mining either cannot scale to large networks, or scale by sacrificing fidelity. Moreover, the specification contains a huge number of low-level intents (e.g., tens of thousands of pairwise reachability), making it hard for operators to read. To this end, this paper presents *NetMiner*, which can mine specification from network configurations, with high scalability, fidelity, and easier to read. The key idea of *NetMiner* is to faithfully simulate the network routing and forwarding behaviors with control plane simulators and data plane verifiers, so as to achieve high fidelity. Meanwhile, *NetMiner* improves the scalability by identifying relevant failure scenarios, and aggregating them to significantly reduce the number of needed simulations. Moreover, *NetMiner* clusters similar low-level intents into a high-level intent, to make the specification more concise and easier to read. Experiments using real configurations from a large cloud service provider and synthetic configurations show that *NetMiner* can mine specification $10\times$ faster, and reduce the number of intents by $100\times$, compared to state-of-the-art tools.**

***Index Terms*—Specification, scalability, fidelity, readability.**

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], [2], [3], [4], [5], [6], [7], generate the configurations from scratch [8], [9], [10], [11], [12], and update the configurations based on the high-level intents [13]. One critical prerequisite of realizing the above vision is the *network specification*, e.g., reachabilities between the prefixes, waypoints that packets should traverse, and the number of link/node failures that can be tolerated.

Received 29 December 2024; revised 16 July 2025; accepted 1 September 2025; approved by IEEE Transactions on Networking Editor L. Fu. Date of publication 24 September 2025; date of current version 2 January 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62272381 and Grant 62572382. The preliminary version was published in [DOI: 10.1109/ICNP59255.2023.10355598]. *(Corresponding author: Peng Zhang.)*

Ning Kang, Peng Zhang, Hao Li, and Sisi Wen are with the Ministry of Education (MOE) Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: kangning2018@foxmail.com; p-zhang@xjtu.edu.cn; hao.li@xjtu.edu.cn; sswen0427@gmail.com).

Chaoyang Ji and Yongqiang Yang are with Huawei Cloud Computing Technology Company Ltd., Beijing 100015, China (e-mail: jichaoyang@huawei.com; yangyongqiang@huawei.com).

Digital Object Identifier 10.1109/TON.2025.3607440

Unfortunately, such a 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 hard 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?

(i) *Scalability problem:* One approach to network specification mining is to simulate or emulate the control plane with control plane simulators or emulators [14], [15], [16], [17] to generate the Routing Information Bases (RIBs) and Forwarding Information Bases (FIBs), and check whether all possible candidate properties (e.g., all-pair reachability) hold with data plane verifiers [18], [19], [20], [21]. 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 must simulate each *failure scenario* and evaluate whether all candidate properties hold under that scenario. For a network with L links, verifying the specification under l-link failures requires enumerating all C_L^l combinations of failed links. In real-world wide-area networks, L can easily reach hundreds (from topology Zoo [22]). For instance, we observed a production WAN topology with $L = 189$ links, which leads to $O(10^6)$ failure scenarios even for $l = 3$ (see Fig. 14). (2) *The cost of each simulation.* To ensure high fidelity, the simulation must not only model Layer-3 routing behavior, but also account for the semantics of lower layers such as Layer-2 (e.g., VLAN, STP) and Layer-1 physical connectivity. This increases the per-simulation cost. In practice, data center networks (DCNs) often exhibit high port density and VLAN complexity, as observed in Google's B4 network [23], Facebook's network [24], and our own deployment. For example, a typical real-world DCN may contain 100 nodes, each with 100 physical or virtual ports, and each port configured with up to 2000 VLANs. Under such settings, even computing the Layer-3 topology requires $O(10^7)$ operations, resulting in a time cost ranging from a few seconds to over a hundred seconds (see Table IV).

2998-4157 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

(ii) *Fidelity problem:* 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], [25]. 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. For example, Tiramisu does not support features like protocol- or port-based packet filters. Most of these models are based on the logical Layer-3 topology, rather than physical Layer-1 topology. Therefore, they may over- or under-estimate the failure tolerance of properties. 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.

(iii) *Readability problem:* Even if we mine the specification from the configurations, the resulting specification contains a large number of intents (or properties). The sheer volume of intents makes it challenging for network operators to comprehend the inherited network. For example, in the DC1 dataset (§ VI-E), with 230 subnets, the number of reachability intents is $\sim 5 \times 10^4$, even larger than the number of configuration lines, which is $\sim 2 \times 10^4$. This is caused by the existing definition of the specification. Such a large number of intents makes the specification hard for human operators to interpret.

The above problems motivate us to ask: *Can we mine the specification from network configurations with both high scalability and fidelity, and readability?* Towards the first question, we make the following observations:

Observation 1: *A limited number of failure scenarios are related to a given property.* In our context, a property refers to data-plane behaviors such as end-to-end reachability, waypointing, or load balancing between subnets at network edge ports [1], [3], [25]. 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, 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 of scenarios. As such, most scenarios can be skipped without simulations. However, in practice, route reflectors are often configured in eBGP [3], causing the forwarding path to diverge from the routing path, which makes it challenging to identify all links (i.e., the forwarding and routing paths) related to the property (see Fig. 6).

Observation 2: *Computing Layer-3 topology dominates the simulation.* To maintain high fidelity, the specification mining must consider failures at the physical and data link layers (i.e., Layer-1 and Layer-2 of the OSI model [26]),[1] instead of only logical failure scenarios at Layer-3 (Network Layer in the OSI model). Note that generating the routing and forwarding tables typically involves two steps: first, computing the Layer-3 topology from the underlying Layer-1 topology and Layer-2 configuration; and second, computing routes over the resulting Layer-3 topology. We observe that computing the Layer-3 topology based on the underlying Layer-1 topology and Layer-2 configuration is surprisingly more costly ($\sim 200\times$ according to our experiments §VI-B) than computing routes based on Layer-3 topology. This high cost is mainly due to the large number of physical ports, each configuring thousands of VLAN, which introduces significant complexity. As such, reducing the cost of computing Layer-3 topology could save considerable time during one simulation.

Observation 3: *A large number of intents in the mined specification share a similar pattern.* For example, consider a typical set of five reachability intents, $s_1 \rightarrow d_1$, $s_1 \rightarrow d_2$, $s_2 \rightarrow d_1$, $s_2 \rightarrow d_2$, $s_3 \rightarrow d_1$. Here, s_1, s_2, s_3 and d_1, d_2 represent non-overlapping source and destination subnets. Clearly, these five intents can be merged into a more concise intent $s_1, s_2, s_3 \rightarrow d_1, d_2$ (*summary intents*) with an exception intent $s_3 \rightarrow d_2$ (*exception intents*), avoiding repeated References to the same subnets. Such a simple pattern occurs frequently in the networks we studied. For example, in the DC1 dataset (§ VI-E), we observe that among 51,631 reachability intents, the source/destination subnets are repeated 227 times on average. Compressing such redundancy can significantly reduce the total number of intents. Even though this example is simple, directly clustering subnets with identical source and destination is not feasible, as this method may still lead to an explosion in the number of resulting intents, reaching up to $O(10^4)$ (see Fig. 21). Therefore, how to balance the number of *summary intents* and *exception intents* is challenging.

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, we make the following contributions.

Contribution 1: General Scenario Aggregation. We propose a *General Scenario Aggregation* method that eliminates the irrelevant failure scenarios. Experiments show that the aggregation method reduces the number of failure scenarios by 2 to 3 orders of magnitude, thereby accelerating the overall intent mining by 5–10$\times$ (see Fig. 13).

Contribution 2: Fast Topology Mapping. We propose a *Fast Topology Mapping* method, which can rapidly generate the Layer-3 topology from the Layer-1 topology. Compared with the baseline approach, generating Layer-3 topology from scratch used by Batfish, our method can reduce the processing time from seconds (192s) to milliseconds (1.423ms) (see Table IV). Finally, based on the reduction in processing the

[1]For simplicity, we use Layer-1 to represent both Layer-1 (physical layer) and Layer-2 (data link layer) of the OSI model, thus the logic of Layer-1 contains physical link aggregation, VLANs, MAC addressing, etc.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

Layer-3 topology, our method achieves an overall speedup ranging from 3× to 357× (see Table V).

Contribution 3: Intent Compression. We introduce the definition of *high-level specification* for describing intents concisely and propose a method named *Intent Compression* to derive such specification. Compared to the *low-level specification*, our method reduces the number of lines by one to two orders of magnitude (see Fig. 19), and the average number of characters per line is reduced by 3× (see Fig. 20) for various properties (e.g., reachability, waypointing), across multiple protocols (e.g., OSPF, BGP), and network types (e.g., WANs, DCNs).

Limitations. *NetMiner* currently cannot mine intents of path preference [1], as it does not model non-best routes, and has limited support for mining traffic-related intents, e.g., the minimum throughput from subnet A to subnet B is 100Mbps. In addition, the fidelity of *NetMiner* depends on the underlying network simulator, e.g., Batfish [14], which may not support all features and devices of various vendors.

Extensions. This paper extends our previous work [27] by improving the readability of the specifications. Specifically, we introduce the definition of *high-level specification*, and propose a method to efficiently derive such *high-level specification*.

II. Motivation

In this section, we motivate specification mining (§II-A), define the problem of *low-level specification* mining (§II-B), and discuss the limitations of existing methods (§II-C). We then explain the necessity for the *high-level specification* (§II-D) and the challenges of deriving it (§II-E).

A. Why Mining Specification

Enabling network changes verification. According to Uptime, an international network advisory organization, their 2021 [28] and 2023 [29] reports indicate that 58% and 45% of network failures, respectively, were caused by configuration changes. Verifying that a configuration achieves its intended effect typically involves analyzing targeted header spaces, e.g., checking reachability between two subnets. However, identifying side effects is harder, because configuration changes may impact seemingly unrelated header spaces and hidden failure scenarios [4]. Moreover, compressing redundant configurations requires verifying the semantic consistency between the original and compressed versions, which usually needs to verify all properties across multiple failure scenarios. Fortunately, as shown in [30], [31], using specifications to analyze configuration changes can reduce configuration-induced errors.

Helping human understanding. Network operators often complain about inheriting an already working legacy network, whose intents are hard to tell [32]. 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. To address this, related works such as Anime [32] and Config2Spec [25] show that specifications assist operators in understanding the network.

TABLE I
Network Properties (s Is Router or Prefix, w Is Router, and d Is Prefix)

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

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. As shown in Uptime's annual reports from 2021 to 2025 [28], [29], [33], [34], human errors accounted for roughly 40% to 63% of network failures. Generally, inferred intents can be used as input for any intent-based networking tool, e.g., automatic configuration synthesis [8], [9], [10], [11], [12]. In addition, automatic configuration mining enables automatic migration, e.g., transparent migration from legacy networks to SDN, other vendor networks [35].

B. Definition of Low-Level Specification

The following formulates the problem of *network specification mining*. The formulation is inspired by Config2Spec [25], 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 consists of links that are up and down, respectively. Node failures can 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 model. 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, $Reachability(s, d) : 3$ means packets sent from source s can reach destination d 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

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

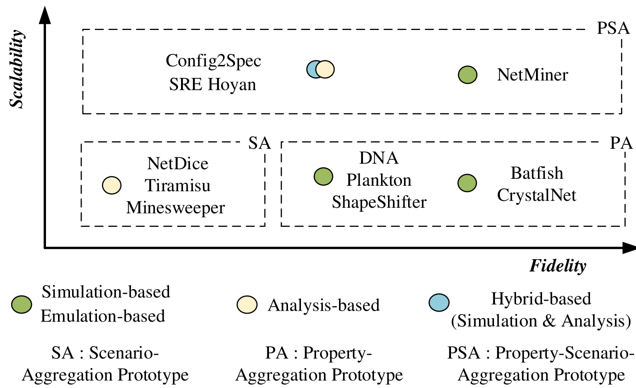


Fig. 1. The tools available for mining specification.

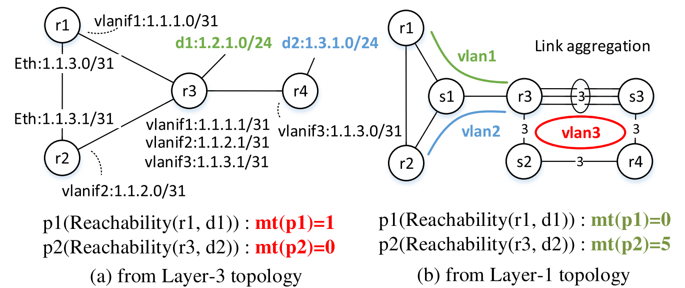


Fig. 2. An example illustrating the differences when mining specifications based on Layer-1 topology and Layer-3 topology. Subnets $d1$ and $d2$ are attached to router $r3$ and router $r4$, respectively. All devices are in the same OSPF domain and the weights of all links are the same.

not accounting for link aggregations). Therefore, we have the following compromised version for specification:

$$Spec_L^C = \{p : min(L, mt(p)) | p \in P^C\} \quad (2)$$

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

Problem 1: Given network configuration C and a bound L of maximum failures, find the specification $Spec_L^C$.

C. Limitations in Mining Low-Level Specification

Low fidelity and scalability. 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 [18], [19], or (2) network behaviors (e.g., network traffic) [8], [32], [36], [37], [38]. 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.

(*i*) *Property-Aggregation Approaches.* Most simulation-based verifiers, e.g., Batfish [14], CrystalNet [15], ShapeShifter [16], Plankton [2] and DNA [17] can generate FIBs and RIBs for a certain scenario, and the data plane verifiers [18], [19], [20] 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.

(*ii*) *Scenario-Aggregation Approaches.* Analysis-based verifiers like Tiramisu [1], NetDice [3] and Minesweeper [6] can check if a 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.

(*iii*) *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 [25] is the closest work to us, which iterates over the simulation-based verifier (Batfish [14]) 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 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.

D. Why High-Level Specification Is Necessary

Many tools [1], [3], [4], [6], [25] express intents using a subnet-to-subnet format, referred to as the *low-level specification*, which results in a large number of intents. The large number of intents, ranging from $\sim 2 \times 10^3$ to $\sim 5 \times 10^4$ lines across two DCN networks and six WANs, with an average of 127 to 139 characters per line (see Experiment VI-E), poses challenges for human comprehension.

To enhance readability and without losing the precision of the *low-level specification*, we propose a dual-view representation format, i.e., the *high-level specification* (syntax outlined in Fig. 12). In the coarse-grained view, *summary intents* abstract intents into a subnet group-to-subnet group format, significantly reducing the number of intents while preserving key semantics and providing operators with a

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

The *low-level specification* (*waypoint*):
(1) 10.10.10.3/28 to 10.10.10.7/28 pass through "frankfurt", and *mt*=3.
(2) 10.10.10.4/28 to 10.10.10.7/28 pass through "frankfurt", and *mt*=3.
(3) 10.10.10.3/28 to 10.10.10.7/28 pass through "paris", and *mt*=2.
(4) 10.10.10.4/28 to 10.10.10.7/28 pass through "paris", and *mt*=2.
(5) 10.10.10.3/28 to 10.10.10.8/28 pass through "frankfurt", and *mt*=3.
(6) 10.10.10.4/28 to 10.10.10.8/28 pass through "frankfurt", and *mt*=3.
(7) 10.10.10.3/28 to 10.10.10.8/28 pass through "paris", and *mt*=2.

The *high-level specification* (*waypoint*):
Summary intents:
(1) subnet cluster 1 $c1$ is {10.10.10.3/28, 10.10.10.4/28}
(2) subnet cluster 2 $c2$ is {10.10.10.7/28, 10.10.10.8/28}
(3) $c1$ to $c2$ have the property ("frankfurt", *mt*=3) ("paris", *mt*=2)
Exception intents:
(3.1) 10.10.10.4/28 to 10.10.10.8/28 pass through "paris", and *mt*=2.

Fig. 3. The *low-level specification* for the network in Fig. 17 and the *high-level specification* for the *low-level specification*.

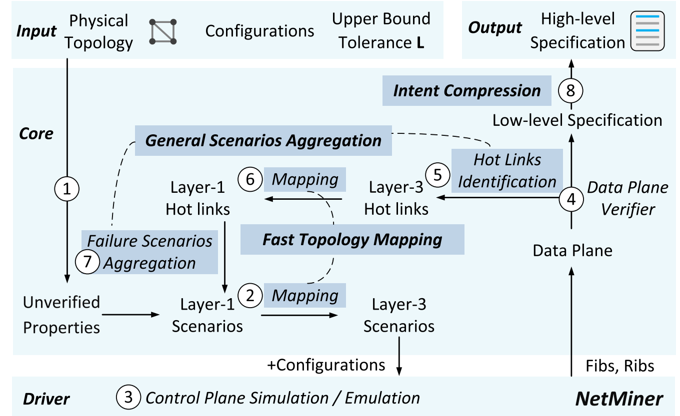


Fig. 4. The workflow of *NetMiner*.

comprehensive overview of the network. In the fine-grained view, *exception intents* address individual intents that do not fit into *summary intents*, ensuring precision by highlighting exceptions. Together, these views enable operators to efficiently understand network intents, with the coarse-grained view offering a broad understanding and the fine-grained view providing the necessary detail to confirm specific cases.

For example, Fig. 3 shows the *low-level specification* used in previous work [25], which requires 7 intents to express the intents and contains redundancy, e.g., the subnet 10.10.10.3/28 is repeated four times. In contrast, as shown in Fig. 3, the *high-level specification* first defines two subnet clusters, $c1$ and $c2$. It then specifies the intent between them: $c1$ to $c2$ has the *summary intent* (*"frankfurt", mt = 3*), meaning that any packet starting from a subnet in $c1$ and reaching a subnet in $c2$ will pass through *"frankfurt"*, even if any two links fail. This approach specifies each subnet only once, eliminates redundancy, and requires just 3 intents instead of 7. Finally, we provide an *exception intent* to this summarization: the intent '10.10.10.4/28 to 10.10.10.8/28 passes through "paris" with *mt = 2*' is not included.

Problem 2: Given a *low-level specification* $Spec$, the goal is to derive a *high-level specification* $\mathcal{H}\text{-}Spec$, consisting of *summary intents* and *exception intents*. The summary intents provide an *abstraction* of $Spec$, while the exception intents refine it. The high-level specification satisfies:

1. (Semantic equivalence) The semantics of $\mathcal{H}\text{-}Spec$, defined as the union of summary intents minus the union of exception intents, is equivalent to $Spec$:

$$\mathcal{H}\text{-}Spec \equiv Spec \tag{3}$$

2. (Minimal size) Among all such specifications, $\mathcal{H}\text{-}Spec$ minimizes the total number of intents:

$$\mathcal{H}\text{-}Spec = \arg \min_{\mathcal{H}\text{-}Spec' \equiv Spec} |\mathcal{H}\text{-}Spec'| \tag{4}$$

where $|\cdot|$ counts both summary and exception intents.

E. Why Deriving High-Level Specification Is Hard

The challenge in deriving the *high-level specification* is to minimize both *summary intents* and *exception intents* simultaneously. This trade-off depends on how subnets are clustered. When more subnets are merged into a single cluster, the *high-level specification* will contain fewer *summary intents* but more *exception intents*, and vice versa (as shown in Experiment VI-E). For example, if each subnet is treated as a separate cluster, the *high-level specification* degenerates into the *low-level specification*.

Therefore, finding a better solution for subnet clustering is challenging. The existing traffic intent compression method, Anime [32], requires operators to specify the label for each subnet, which increases the operator's burden and may not lead to a better solution. Additionally, if we directly cluster source subnets with identical destinations, the number of *summary intents* may not decrease compared to the *low-level specification*. For example, in the case above, all subnets would be treated as individual clusters because the destinations of source subnets 10.10.10.3/28 and 10.10.10.4/28 are not exactly the same. Finally, enumerating all possible subnet clustering combinations leads to exponential time complexity [39].

Thus, *NetMiner* aims to find a solution of subnet clusters that minimizes the number of *summary* and *exception intents*.

III. Overview

In this section, we first show the workflow of *NetMiner* (§III-A). Then we explain how to achieve high scalability with *General Scenario Aggregation (GSA)* (§III-B), high fidelity with *Fast Topology Mapping (FTM)* (§III-C), and high readability with *Intent Compression (IC)* (§III-D).

A. Workflow of NetMiner

The workflow of *NetMiner* is shown in Fig 4, which takes as the input the network configuration C, *NetMiner* outputs the formal specification $Spec^c_L$ 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*)

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

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 *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 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.

Afterward, *NetMiner* generates the *low-level specification* with a large number of intents. *NetMiner* then compresses it to produce the *high-level specification* with a concise format (⑧).

We highlight three 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. Third, *NetMiner* must identify a better solution from all combinations of subnet clusters that minimizes both the number of *summary* and *exception intents* in the *high-level specification*. *NetMiner* designs three key modules, *General Scenario Aggregation (GSA)*, *Fast Topology Mapping (FTM)* and *Intent Compression (IC)* for addressing the above three challenges. We present these three modules with concrete examples in the following sections.

B. General Scenario 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.

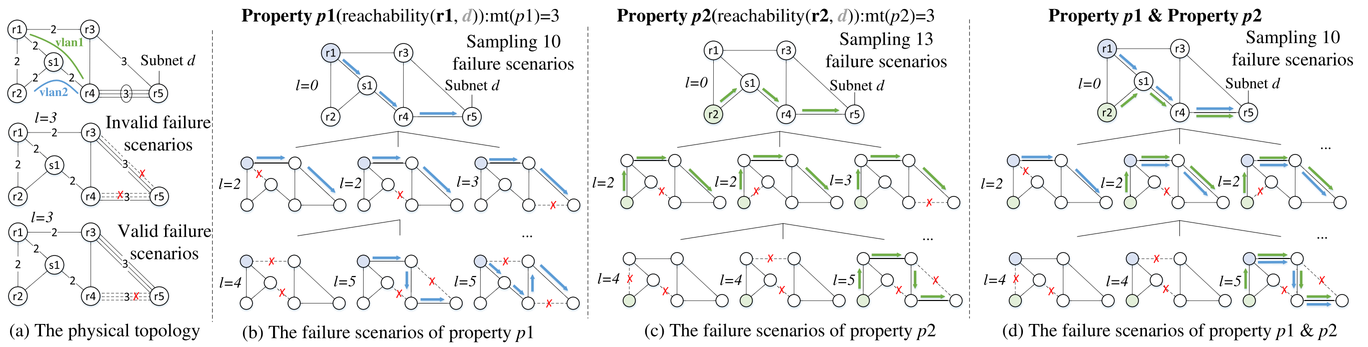
We observe that the vendor-dependent behaviors, i.e., route propagation and best route selection, are already handled after the simulators compute the routes, i.e., routing table. This reveals the chance to trace forwarding paths and routing paths, purely from the rules of the routing table, e.g., the routing 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. 5(a) as an example, and consider two reachability properties $p1$ ($r1 \rightarrow d$) and $p2$ ($r2 \rightarrow d$), with a bound $L = 3$ on failure tolerance. Fig. 5(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 14 shows that failure scenarios aggregation can reduce the total number of simulations by 1-2 orders of magnitude (see § IV-A for details).

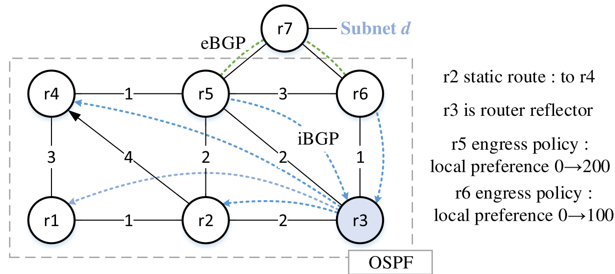
A walk-through example. We show an example in Fig. 6(a) to illustrate how we identify the *hot links*, and hence reduce the failure scenarios. We consider the reachability property $p1$ from device $r1$ to subnet d. (1) For forwarding paths, we start from device $r1$, and find the next hop to d is $r5$ 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. 6(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. 6(c), if we don't consider routing recursion, the link $r2-r5$ would be ignored and actually affect the forwarding behavior.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

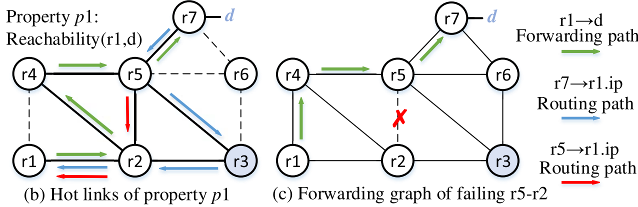


(a) The physical topology (b) The failure scenarios of property $p1$ (c) The failure scenarios of property $p2$ (d) The failure scenarios of property $p1$ & $p2$

Fig. 5. 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. 6. 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.

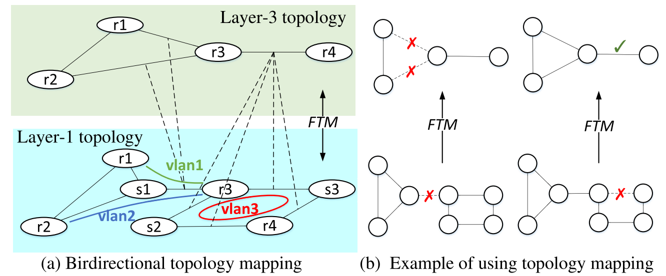
C. Fast Topology Mapping

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



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

Fig. 7. An example of *Fast Topology Mapping*.

in Fig. 7(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 paths $r1-s1-r3$ and $r2-s1-r3$ of these two links at the Layer-1 topology are all blocked, so both links failed. For another example, when $s3-r3$ in Layer-1 is failed, we determine that $r3-r4$ in Layer-3 will be affected, and find that there is another path, i.e., $r3-s2-r4$ in Layer-1. Then, no Layer-3 links are affected.

D. Intent Compression

NetMiner avoids the enumeration of subnet clusters by using the heuristic method *Intent Compression (IC)*. First, we formally convert the *low-level specification* into the *property matrix*, as shown in Fig. 8 (left), for the purpose of generally processing properties.

Then, for a given source subnet, all related intents, where this subnet serves as the source, form a row vector in property matrix. The difference between vectors of two subnets quantifies the similarity between the two subnets. We observe that this similarity is directly related to *summary* and *exception intents* in the specification. If the distance between two subnets is small, merging them results in fewer *exception intents* in the *high-level specification*. However, subnets with smaller distances are relatively few, which leads to a higher number of *summary intents*, and vice versa. We then extract the difference between subnets as a distance matrix, as shown in Fig. 8 (right). Based on this, we use the *hierarchical clustering* method [40], which iteratively clusters two elements with the minimal distance until only one cluster remains, generating a *dendrogram* as shown in Fig. 9. We then enumerate the distance threshold to derive the *high-level specification*, avoiding the enumeration of cluster combinations.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

$$\begin{array}{c} \\ s_1 \\ s_2 \\ \\ s_3 \end{array}\begin{pmatrix} \overset{d_1}{\varnothing} & \overset{d_2}{\varnothing} & \overset{d_3}{\varnothing} \\ \varnothing & \begin{matrix}(w_1,3)\\(w_2,2)\end{matrix} & \begin{matrix}(w_1,2)\\(w_2,2)\end{matrix} \\ \varnothing & \begin{matrix}(w_1,3)\\(w_2,2)\end{matrix} & (w_1,3) \end{pmatrix} \Rightarrow \begin{array}{c|ccc} & s_1 & s_2 & s_3 \\ \hline s_1 & 0 & 4 & 4 \\ s_2 & 4 & 0 & 1 \\ s_3 & 4 & 1 & 0 \end{array}$$

Fig. 8. The property matrix (left) and the distance matrix (right) for the *low-level specification* in Fig. 12. For a better explanation, assume that there are two additional subnets 10.10.10.2/28 and 10.10.10.7/28. Here, s_1=10.10.10.2/28, s_2=10.10.10.3/28, s_3=10.10.10.4/28, d_1=10.10.10.6/28, d_2=10.10.10.7/28, d_3=10.10.10.8/28. w_1 is *frankfurt* and w_2 is *paris*. For the property matrix, the integer number is the tolerance. For the distance matrix, the integer number is the distance between two subnets.

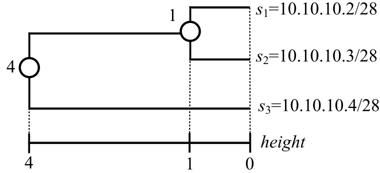


Fig. 9. The *dendrogram* generated by *hierarchical clustering* for source subnets in Fig. 8. The height value represents the distance at which two clusters are merged.

In real cases, the number of lines in the *high-level specification* is reduced by one to two orders of magnitude, as shown in Fig. 19.

A walk-through example. Based on the *dendrogram* shown in Fig. 9, we can effectively derive subnet clusters. For example, when $T_{\mathcal{M}} = 0$, the tree is cut into three nodes, corresponding to $\{s_1\}, \{s_2\}, \{s_3\}$. For $T_{\mathcal{M}} = 1 \sim 3$, the clusters are $\{s_1\}, \{s_2, s_3\}$. When $T_{\mathcal{M}} \geq 3$, the clusters are $\{s_1, s_2, s_3\}$. We then generate the *high-level specification* and count the number of intents (or lines) for each solution of subnet clusters. We observe that when $T_{\mathcal{M}} = 1 \sim 3$, the *high-level specification* has the minimum number of intents: 4, as shown in Fig. 3.

IV. Mining Low-Level Specification From Configurations

A. General Scenario 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.

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. Meanwhile, *NetMiner* resolves the path that those routes are propagated. Then, *NetMiner* classifies all links on the forwarding and routing paths as "hot", and computes relevant failure scenarios by failing one of those *hot links* each time.

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** $\mathcal{H}$

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 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

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 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 $Info(s,d)$.

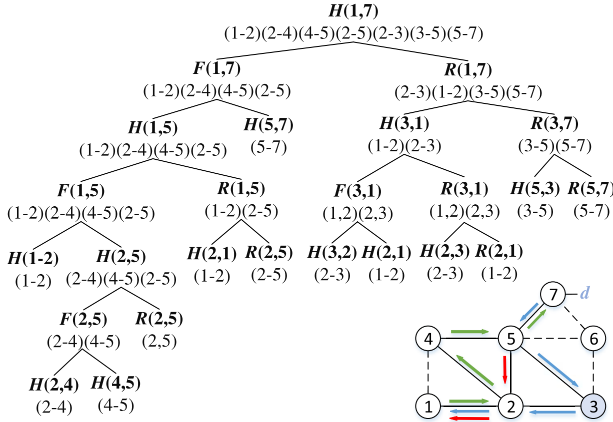


Fig. 10. The process of Alg. 1 running on the network of Fig. 6. We simplified the function parameters in the graph. We omit the leaf nodes in the graph, which are the functions F and R.

(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 10 shows the tree when using Alg. 1 to compute *hot links* for the previous example (Fig 6). 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 links $\mathcal{H}$ returned by Alg. 1 contains all hot links for p(s, d) on f.

Proof: For clarity, we only show the proof sketch here.[2] 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. This 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).

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

NetMiner is extensible to mine any property that can be inferred by the best routes from 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 recompute 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.[3] 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 11, R2-Eth2 and S1-Eth4 are two connected Layer-2 ports.

[2]The full proof is available at https://github.com/916267142/NM-Prove

[3]One viable method of obtaining $\mathcal{L}1$ is to use the SNMP protocol to read the Physical Topology MIB of each device [41].

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

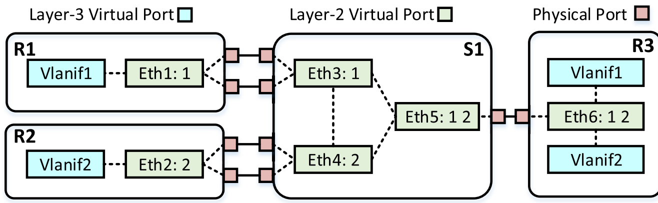


Fig. 11. An illustration of Layer-3 topology construction for Fig 7. Here, dashed and solid links represent the virtual links we create and the real links, respectively.

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 11, itNetMiner constructs a Layer-3 link (R2-Vlanif2, R3-Vlanif2) since their frames can traverse a path R2-Eth2 $\rightarrow$ S1-Eth4 $\rightarrow$ S1-Eth5 $\rightarrow$ R3-Eth6.

During the construction of Layer-3 topology, *NetMiner* also computes two maps. For each $l^3 \in L3$, let $Paths(l^3)$ be the set of all physical paths for l^3. For example, in Fig 7, $Paths((r3, r3)) = \{r3 - s3 - r4, r3 - s2 - r4\}$. 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:

$$Links(l^1) = \{l^3 | \exists path \in Path(l^3), l^1 \in path\} \quad (5)$$

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:

- $M^{1\rightarrow 3} : \mathcal{L}1 \rightarrow 2^{\mathcal{L}3}$, 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.
- $M^{3\rightarrow 1} : \mathcal{L}3 \rightarrow 2^{\mathcal{L}1}$, 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. (6) to compute the Layer-3 failed links $M^{1\rightarrow 3}(L^1)$, without re-computing the Layer-3 topology using simulators (e.g., Batfish).

$$M^{1\rightarrow 3}(L^1) = \{l^3 \in Links(l^1) | l^1 \in L^1, \nexists path \in Path(l^3), path \wedge L^1 = \varnothing\} \quad (6)$$

The time complexity for computing Eq. (6) 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. (7) to identify the corresponding Layer-1 *hot links* $M^{3\rightarrow 1}(l^3)$.

$$M^{3\rightarrow 1}(l^3) = \{l^1 \in \mathcal{L}1 | \exists path \in Paths(l^3), l^1 \in path\} \quad (7)$$

The time complexity for computing Eq. (7) is $O(M)$.

This approach may overestimate 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 overestimation, and therefore have not considered 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.

Trimming based on enumeration analysis. 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. From Low-Level Specification to High-Level Specification

A. Definition of High-Level Specification

Fig. 12 shows the syntax of the *high-level specification* $\mathcal{H}$*-Spec*, which consists of a set of specifications for each type of property.[4] Each property-specific specification $\mathcal{H}$*-Spec*$_{type}$ is defined as follows:

(i) **The *summary intent***, denoted as *intent*$_{summary}$, consists of two types of statements, *statement*$_s$ and *statement*$_p$, *statement*$_s$ defines the subnet cluster, e.g., $c_1 : \{10.10.10.3/28, 10.10.10.4/28\}$; *statement*$_p$ defines the properties that hold among subnet clusters, where each property is specified by the source subnet cluster, the destination subnet cluster, and the attributes (tag and tolerance). Depending on the property type, the tag may represent the device name for waypointing, the number of disjoint paths for load balancing, etc. For example, one *summary intent* in Fig. 12 can be expressed as (c_1, c_2, $\{(\text{"}frankfurt\text{"}, 3), (\text{"}paris\text{"}, 2)\}$).

(ii) **The *exception intent***, denoted as *intent*$_{exception}$, is the same as the intent in the *low-level specification*. The semantics are as follows: if *intent*$_{exception}$ is included in *intent*$_{summary}$, then $\mathcal{H}$*-Spec*$_{type}$ excludes this. Conversely, it means $\mathcal{H}$*-Spec*$_{type}$ includes this. We can use additional information for annotation, but for simplicity, we unify them as *exception intents* here. For example, the *exception intent* in Fig. 12 can be expressed as (10.10.10.4/28, 10.10.10.8/28, *"paris"*, 2).

[4]We design a separate specification to provide a better understanding, allowing operators to flexibly choose the specification corresponding to the properties they are interested in.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

$$\mathcal{H}\text{-}Spec \leftarrow \{\mathcal{H}\text{-}Spec_{type}\} \tag{8}$$
$$\mathcal{H}\text{-}Spec_{type} \leftarrow \{intent_{summary} | intent_{exception}\} \tag{9}$$
$$intent_{summary} \leftarrow statement_s \mid statement_p \tag{10}$$
$$statement_s \leftarrow \{subnet\} \tag{11}$$
$$statement_p \leftarrow (statement_s, statement_s, \{attribute\}) \tag{12}$$
$$attribute \leftarrow (tag, tolerance) \tag{13}$$
$$intent_{exception} \leftarrow (subnet, subnet, tag, tolerance) \tag{14}$$
$$type \leftarrow \textbf{reachability} \mid \textbf{waypoint} \mid \textbf{load balancing} \tag{15}$$
$$tag \leftarrow \textbf{string} \mid \textbf{int} \tag{16}$$
$$tolerance \leftarrow \textbf{int} \tag{17}$$

Fig. 12. Simplified syntax for the *high-level specification*.

B. Intent Compression

NetMiner uses *Intent Compression (IC)* to derive the *high-level specification* $\mathcal{H}$*-Spec*, which consists of $\mathcal{H}$*-Spec*$_{type}$, so in the following, we will explain how to derive $\mathcal{H}$*-Spec*$_{type}$

Step 1. Quantifying the distances. The first step is to quantify the distances among different subnets, such that we can use the distances to cluster subnets. To achieve this, we represent the *low-level specification* with a property matrix $\mathcal{M}$ defined as follows.

$$\mathcal{M}(s,d) = \{(tag,\ mt)|(s,d,tag) : mt \in \mathcal{L}\text{-}Spec\} \tag{18}$$

Here, s and d are the source and destination subnets. The left of Fig. 8 shows a property matrix.

For the source subnet (i.e., the row axis), we use $\mathcal{M}(s,:)$ to represent the intents in which the source subnet s appears:

$$\mathcal{M}(s,:) = (\mathcal{M}(s,0), \mathcal{M}(s,1), \ldots, \mathcal{M}(s,N)) \tag{19}$$

For two source subnets i and j, their distance is defined as the difference of the elements in the corresponding rows of their property matrices:

$$\mathcal{D}^{row}_{\mathcal{M}}(i,j) = \sum_{k \in Col(\mathcal{M})} |\mathcal{M}(i,k) \triangle \mathcal{M}(j,k)|, \tag{20}$$

where $\triangle$ is the symmetric difference of sets, e.g., $\{a,b\} \triangle \{b,c\} = \{a,c\}$.

Intuitively, the distance between two subnets corresponds to the number of *exception intents* after clustering these two subnets. Moreover, the number of subnets with a certain distance decides the number of subnet clusters, which in turn decides the number of *summary intents*.

Then, we obtain the $\mathcal{D}^{row}_{\mathcal{M}}$ matrix as shown in Fig. 8 (right). The distance matrix for destination subnet $\mathcal{D}^{col}_{\mathcal{M}}$ can be generated in the same way.

Step 2. Calculating dendrogram. After extracting features of subnet, there are some clustering methods to identify subnet clusters. Since we do not know the number of clusters beforehand, many methods, such as K-means [42], do not apply. Other approaches like DBSCAN [43] and OPTICS [44] do not require the number of clusters as input, but rely on some threshold parameters (e.g., distance threshold), which is difficult to estimate. Mean Shift [45] and Affinity Propagation [46], on the other hand, generate clusters in one shot, making it difficult to adjust if the clustering result is not good.

For the above reasons, we use *agglomerative clustering*, a bottom-up *hierarchical clustering* approach, which has been successfully applied to clustering cloud intent [47] and traffic [32]. Specifically, *NetMiner* uses Single-Linkage [40], one variant of *agglomerative clustering*, to determine which two clusters should be merged: at each iteration, it selects the two clusters that minimize the maximum distance between any two subnets i and j, calculated by the matrix $\mathcal{D}^{row}_{\mathcal{M}}(i,j)$ for source subnets (and $\mathcal{D}^{col}_{\mathcal{M}}(i,j)$ for destination subnets). The output of the *hierarchical clustering* is a *dendrogram* (tree) of subnets, where the leaf nodes represent the subnets, and the non-leaf nodes represent the clusters. Each non-leaf node has a value termed height, which represents the distance between the two entities merged at that node, as shown in Fig. 9. To this end, a static threshold can be used: all nodes that are just below this threshold correspond to distinct clusters [47].

Step 3. Deriving high-level specification. After calculating the *dendrogram* for source and destination subnets respectively, we derive $\mathcal{H}$*-Spec*$_{type}$ as follows. For a given property *type*, we enumerate the distance threshold $T_{\mathcal{M}}$ starting from $T_{\mathcal{M}} = 0$ to relatively large upper bound. We derive the *high-level specification* and calculate its lines corresponding to $T_{\mathcal{M}}$. For a given $T_{\mathcal{M}}$, we separately cluster the source and the destination subnets. For source subnets, we get the clusters from the *dendrogram*. After it, all clusters of source subnet and destination subnet form $statement_s$. For $statement_p$, we enumerate pairs between all $statement_s$. For a source $statement_s$ and a destination $statement'_s$, we generate the attributes set $\mathcal{A} = \{\mathcal{M}(s,d) \mid s \in statement_s \wedge d \in statement'_s\}$. We select the attribute set $a \in \mathcal{A}$ that appears most frequently in $\mathcal{A}$, and then $(statement_s, statement'_s, a)$ forms $statement_p$. Next, all $statement_s$ and $statement_p$ form *summary intents*. We then take the difference between *summary intents* and the *low-level specification* to obtain the *exception intents*. Finally, *summary* and *exception intents* together form $\mathcal{H}$*-Spec*$_{type}$.

The time complexity of *Intent Compression* is $O(N^2)$, where N is the number of intents, as proven in [48].

Our *Intent Compression* is primarily designed for intents as shown in Table I, and is not applicable to intents involving traffic-related intents, because our enumeration method cannot model continuous variables.

C. Optimizations

Merging source and destination statements. After deriving *the high-level specification*, we further combine all source and destination subnets by splitting them into new subnets. Then, if two new subnets belong to the same source statement and destination statement, they are clustered into one new $statement_s$; otherwise, they are assigned to separate $statement_s$.

Reducing the threshold enumeration. Since *summary intents* are non-decreasing with an increasing distance threshold, while *exception intents* are non-increasing with an increasing distance threshold.[5] Then, the sum of *summary intents* and *exception intents* initially decreases with distance. After reaching the critical point, it will start to increase.

[5]This is evident because for two thresholds T_1 and T_2, where $T_1 > T_2$. It is easy to use proof by contradiction to deduce that the number of subnet clusters for T_1 will be less than or equal to that for T_2. Consequently, the *summary intents* of T_1 will be less than or equal to those of T_2.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

Therefore, we start enumerating the thresholds from 0, and when the total lines of intents begin to show an increasing trend, we stop the enumeration (as shown in Experiment VI-E).

VI. Evaluation

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

Mining the low-level specification:

- 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. (§ VI-A)
- How do *General Scenario 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. (§ VI-B)
- Can *NetMiner* support various configuration features without fidelity loss? Experiments on real configurations show that *NetMiner* avoids some incorrect results returned by Config2Spec. (§ VI-C)

Deriving the high-level specification:

- Is the *high-level specification* derived by *NetMiner* consistent with operator intent? We validate the *high-level specification* with both real and synthetic network configurations. (§ VI-D)
- How readable is *Intent Compression (IC)*? We show that the *high-level specification* reduces the number of lines by one to two orders of magnitude, compared to the *low-level specification*. (§ VI-E)

Implementation. We implement *NetMiner* with $8k$ lines of C++ code and an extra $2k$ lines to re-implement Delta-net [19]. We use Batfish[6] [14] 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 [19] 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 [18].

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 [49]. We use Config2Spec [50] 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 [25]. The network consists of 10 routers and 18 links.
3. Synthesized configurations for three WAN networks running BGP or OSPF, from Config2Spec [25]. 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 of Mining Low-Level Specification

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.

Fig 13 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 of Mining Low-Level Specification

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

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

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

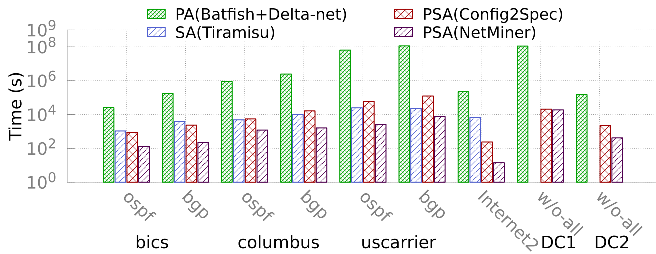


Fig. 13. Time to mine specifications for reachability, waypoint, and load balancing. Here, since PA cannot run to complete within a day, we report the estimated time based on the average time for a single failure scenario and the total number of failure scenarios.

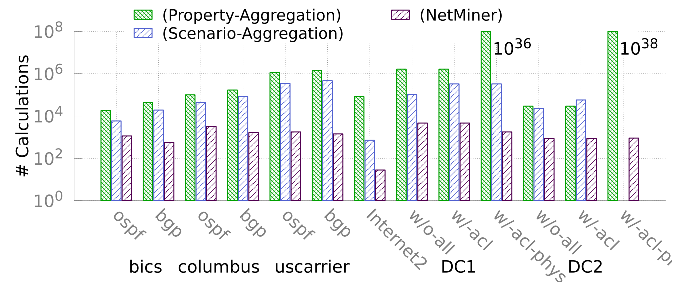


Fig. 14. The number of simulations (for PA and *NetMiner*) or verifications (for SA) of different methods.

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

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

TABLE V

TIME COMPARISON FOR CHECKING ALL SCENARIOS, INCLUDING COMPUTING THE LAYER-3 TOPOLOGY AND SIMULATING THE CONTROL PLANE

Datasets		Batfish	NetMiner	Speedup
DC1	$l = 1$	6540s	1860s	3.5
	$l = 2$	13080s	3720s	3.5
	$l = 3$	19314s	5493s	3.5
DC2	$l = 1$	57972s	162s	357.4
	$l = 2$	115944s	324s	357.1
	$l = 3$	173916s	487s	356.9

General Scenario 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 Fig 14, *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 15. 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. *FTM* is 5 to 6 orders of magnitude faster than Batfish in Table IV.

To fairly demonstrate the performance improvement of *FTM*, we compare the total time to check all scenarios using *Batfish* and *NetMiner* on two DCN datasets. On DC1, *NetMiner* is consistently 3.5× faster, while on DC2, the speedup ranges from 356.9× to 357.4×.

C. Fidelity of Low-Level Specification

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.

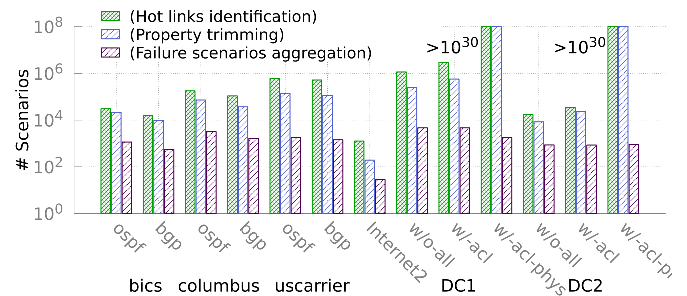


Fig. 15. Comparison of the number of failure scenarios, after applying hot links identification, property trimming, and failure scenarios aggregation.

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

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

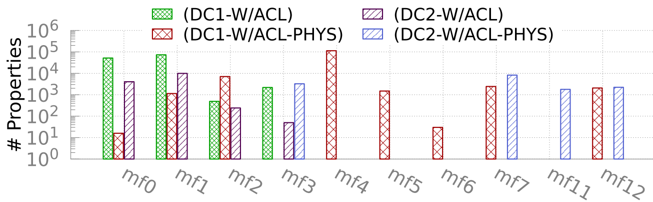


Fig. 16. Distribution of the number of properties for different tolerances between considering physical link aggregation (W/ACL-PHYS) and not considering physical link aggregation (W/ACL).

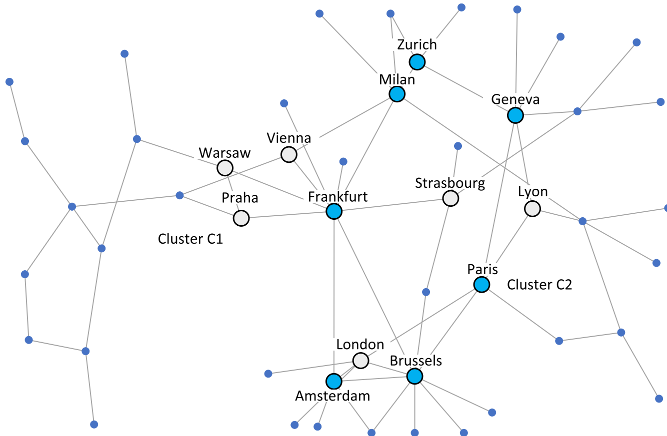


Fig. 17. The subnet clusters for Bics-BGP dataset from real topology when deriving the *high-level specification* for reachability. Each device only configures one edge subnet. So we use the device to represent the subnet on it. The subnets of light (dark) nodes belong to subnet cluster $c1$($c2$).

Real failure model. As shown in Fig 16, 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], [4], [5], [6], one may underestimate 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.

D. Validity of High-Level Specification

Case study 1. We illustrate that *IC* has physical significance and can help operators diagnose faults in annotations. Specifically, annotations such as “external_om” or “pod_cascaded_storage” is configured on ports to explain the purposes of their corresponding subnets. We say an annotation is consistent if all subnets with the annotation belong to a single subnet cluster calculated by *IC*. Then, we use the following metric of *consistency ratio* to measure how well the subnet cluster computed by *IC* aligns with the real purpose.

$$\textit{consistency ratio} = \frac{\#\ \textit{of consistent annotations}}{\#\ \textit{of all annotations}}$$

For our DC1 dataset, 40 out of these 42 annotations are consistent, while for 2 annotations, their subnets span multiple subnet clusters. This results in a consistency ratio of $40/42 = 95.2\%$. Among 2 inconsistent annotations, 24 subnets are annotated with a description of “pod_cascaded_storage”, a type of storage service in the cloud datacenter. 14 of these subnets fall in one cluster and the other 10 fall in another

The *high-level specification (reachability)*:

(1) subnet cluster $c1$: Warsaw, Vienna, Praha, Strasbourg, Lyon, London
(2) subnet cluster $c2$: Frankfurt, Milan, Zurich, Geneva, Paris, Brussels, Amsterdam
(3) c1 to c2 have the property (*mt*=2)
(4) c2 to c2 have the property (*mt*=3)
(5) c1 to c1 have the property (*mt*=2)
(6) c2 to c1 have the property (*mt*=2)

Fig. 18. The *low-level specification* and the *high-level specification* of reachability intents about the subnet of light and dark nodes in Fig. 17.

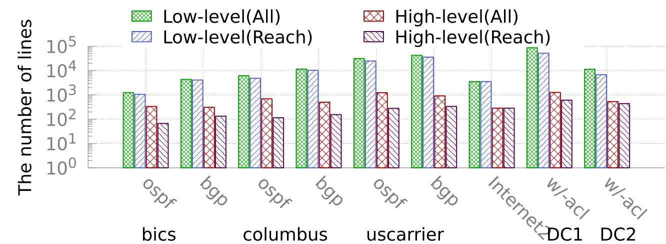


Fig. 19. Comparison of the number of lines between the *low-level specification* and the *high-level specification*. ‘Reach’ represents a specification that contains only reachability, while ‘All’ represents a specification that includes reachability, waypoints, and load balancing.

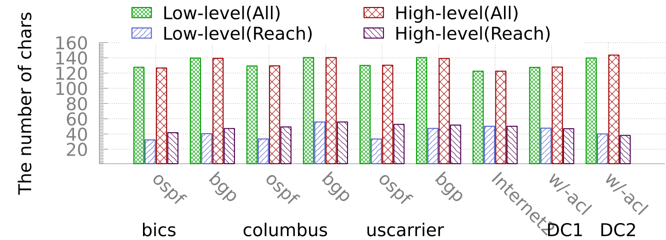


Fig. 20. Comparison of the average number of characters per line between the *low-level specification* and the *high-level specification*. ‘Reach’ represents a specification that contains only reachability, while ‘All’ represents a specification that includes reachability, waypoints, and load balancing.

one. We confirm with the operators that this is an error in the annotation since these two groups of subnets actually provide different services. For these 24 subnets, 14 of subnets are responsible for high-frequency access data, while the other 10 subnets handle low-frequency access data. Finally, our *high-level specification* has only 608 lines, much smaller than the original *low-level specification*, which has 51,631 lines.

Case study 2. In the above, we consider data center networks whose topologies are quite regular. To study whether *IC* also works for networks with irregular topologies, we experiment with the Bics-BGP dataset synthesized by [25] on a real WAN topology. The synthesized configurations ensure a simple policy of all-pair reachability, and thus the values of failure tolerance are entirely determined by the topology. Fig. 17 shows cluster 1 (2) marked by light (dark) node. For the *low-level specification*, we need $13 \times 12 = 156$ intents to describe the reachability between the 13 subnets included in the light and dark nodes. In contrast, for the *high-level specification*, we only need 6 intents, as shown in Fig. 18. Finally, with 4093 lines of the *low-level specification*, *IC* derives 134 lines of the *high-level specification*.

Semantic equivalence. The intent compression method should ensure that the *high-level specification* and the *low-level specification* are semantically equivalent. To check whether

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

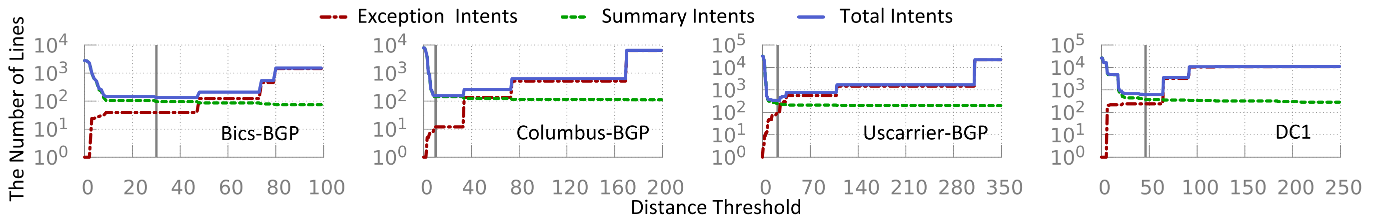


Fig. 21. The trend curve of the number of lines of *exception intents*, *summary intents* and total lines of the *high-level specification* with respect to the increasing *distance threshold*. The gray dashed line represents the threshold selected by *Intent Compression*, which is the minimum value. Since the OSPF dataset and DC2 exhibit the same trend, we omit them for clarity.

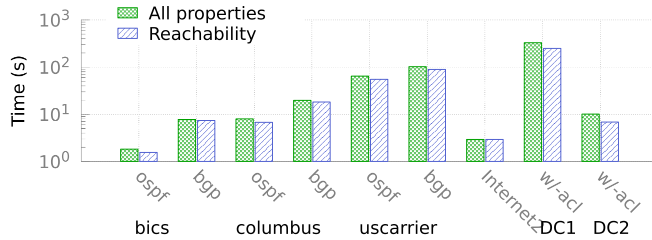


Fig. 22. The calculation time of *Intent Compression (IC)*.

this requirement is met, we unfold the *high-level specification* into a *low-level specification*, and compare it to the original *low-level specification*. We confirm that they match exactly on all of our datasets.

E. Readability of High-Level Specification

Number of lines. As shown in Fig. 19, the number of lines of the *high-level specification* is one to two orders of magnitude smaller than that of the *low-level specification*, for different types of properties (reachability, waypoint, load balancing), protocols (OSPF, BGP), and networks (WAN and DCN).

Number of characters. As shown in Fig. 20, the average number of characters per line in the *high-level specification* is 3× less than that of the *low-level specification*.

Threshold selection. In this experiment, we show how *IC* can find a threshold that minimizes the total number of intents. Fig. 21 shows that as the *distance threshold* increases, the number of lines of *summary intents* and *exception intents* is non-increasing and non-decreasing, respectively. Therefore, we can always find a threshold that minimizes their sum, i.e., the total number of intent, as shown by the dashed lines.

F. Performance of Deriving High-Level Specification

Fig. 22 shows the computation time of *Intent Compression*. For all different types of properties, protocols, and networks, *Intent Compression* finishes in minutes. This indicates that the time required to generate the high-level specification is negligible compared to the time spent generating *low-level specifications*.

VII. Conclusion

We propose *NetMiner*, a network specification mining tool with high scalability, fidelity and readability. *NetMiner* is built upon pure simulation-based verifiers, so as to abstract away all the vendor-specific models. *NetMiner* designs (1) a *General Scenario Aggregation* to analyze the property-related scenarios only, and (2) a *Fast Topology Mapping* that incrementally transforms Layer-1 topology to Layer-3 topology, (3) an *Intent Compression* that compresses the intents in the specification into a concise format. 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, and reduce the number of lines in the specification to the hundreds.

References

[1] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast multilayer network verification," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 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 *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 953–967.
[3] S. Steffen, T. Gehr, P. Tsankov, L. Vanbever, and M. Vechev, "Probabilistic verification of network configurations," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 750–764.
[4] P. Zhang, D. Wang, and A. Gember-Jacobson, "Symbolic router execution," in *Proc. ACM SIGCOMM Conf.*, Aug. 2022, pp. 336–349.
[5] F. Ye et al., "Accuracy, scalability, coverage: A practical configuration verifier on a global WAN," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 599–614.
[6] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 155–168.
[7] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proc. ACM SIGCOMM Conf.*, Aug. 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 *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 328–341.
[9] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "Network-wide configuration synthesis," in *Proc. 29th Int. Conf. Comput. Aided Verification (CAV)*. Heidelberg, Germany: Springer, Jul. 2017, pp. 261–281.
[10] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "NetComplete: Practical network-wide configuration synthesis with autocompletion," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 579–594.
[11] K. Subramanian, L. D'Antoni, and A. Akella, "Synthesis of fault-tolerant distributed router configurations," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 2, no. 1, pp. 1–26, Jan. 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 *Proc. ACM SIGPLAN Int. Conf. Object-Oriented Program., Syst., Lang., Appl.*, Oct. 2016, pp. 765–780.
[13] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: Incrementally synthesizing policy-compliant and manageable configurations," in *Proc. 16th Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2020, pp. 482–495.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.

[14] A. Fogel et al., "A general approach to network configuration analysis," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 469–483.
[15] H. H. Liu et al., "CrystalNet: Faithfully emulating large production networks," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 599–613.
[16] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Abstract interpretation of distributed network control planes," in *Proc. ACM Program. Lang.*, 2019, vol. 4, no. POPL, pp. 1–27.
[17] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *Proc. USENIX NSDI*, 2022, pp. 601–615.
[18] P. Zhang, X. Liu, H. Yang, N. Kang, Z. Gu, and H. Li, "APKeep: Realtime verification for real networks," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 241–255.
[19] A. Horn, A. Kheradmand, and M. R. Prasad, "Delta-Net: Real-time network verification using atoms," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 735–749.
[20] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 887–900, Apr. 2016.
[21] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Aug. 2012, pp. 49–54.
[22] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
[23] S. Jain et al., "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, Aug. 2013, pp. 3–14.
[24] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *Proc. Opt. Interconnects Conf.*, May 2013, pp. 49–50.
[25] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Config2Spec: Mining network specifications from network configurations," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2020, pp. 969–984.
[26] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed., Boston, MA, USA: Pearson, 2010.
[27] N. Kang, P. Zhang, H. Li, S. Wen, C. Ji, and Y. Yang, "Network specification mining with high fidelity and scalability," in *Proc. IEEE 31st Int. Conf. Netw. Protocols (ICNP)*, Oct. 2023, pp. 1–11.
[28] Uptime Institute. (2021). *Annual Outage Analysis Report 2021*. [Online]. Available: https://uptimeinstitute.com/resources/research-and-reports/annual-outage-analysis-2021
[29] (2023). *Annual Outage Analysis Report 2023*. [Online]. Available: https://uptimeinstitute.com/resources/research-and-reports/annual-outage-analysis-2023
[30] X. Xu et al., "Relational network verification," in *Proc. ACM SIGCOMM Conf.*, Aug. 2024, pp. 213–227.
[31] M. Brown, A. Fogel, D. Halperin, V. Heorhiadi, R. Mahajan, and T. Millstein, "Lessons from the evolution of the batfish configuration analysis tool," in *Proc. ACM SIGCOMM Conf.*, Sep. 2023, pp. 122–135.
[32] A. Kheradmand, "Automatic inference of high-level network intents by mining forwarding patterns," in *Proc. Symp. SDN Res.*, Mar. 2020, pp. 27–33.
[33] *Annual Outage Analysis Report 2024*, Uptime Institute, Seattle, WA, USA, 2024. [Online]. Available: https://uptimeinstitute.com/resources/research-and-reports/annual-outage-analysis-2024
[34] (2025). *Annual Outage Analysis Report 2025*. [Online]. Available: https://uptimeinstitute.com/resources/research-and-reports/annual-outage-analysis-2025
[35] L. Huang and L. Lu, "Segmentation of ischemic stroke lesion based on long-distance dependency encoding and deep residual U-Net," *J. Comput. Appl.*, vol. 41, no. 6, p. 1820, 2021.
[36] C. J. Anderson et al., "NetKAT: Semantic foundations for networks," *Acm SIGPLAN Notices*, vol. 49, no. 1, pp. 113–126, 2014.
[37] R. Soulé et al., "Merlin: A language for provisioning network resources," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol.*, Dec. 2014, pp. 213–226.
[38] C. Prakash et al., "PGA: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 29–42, 2015.
[39] M. Aigner, "A characterization of the bell numbers," *Discrete Math.*, vol. 205, nos. 1–3, pp. 207–210, Jul. 1999.
[40] F. Nielsen, "Hierarchical clustering," in *Introduction to HPC With MPI for Data Science*. Cham, Switzerland: Springer, 2016, pp. 195–211.
[41] A. Bierman and K. Jones, *Physical Topology Mib*, document RFC2922, 2000.
[42] J. MacQueen et al., "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.
[43] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 226–231.
[44] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, Jun. 1999.
[45] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.
[46] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, Feb. 2007.
[47] W. Pang, S. Panda, J. Amjad, C. Diot, and R. Govindan, "CloudCluster: Unearthing the functional structure of a cloud service," in *Proc. 19th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2022, pp. 1213–1230.
[48] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *Comput. J.*, vol. 16, no. 1, pp. 30–34, Jan. 1973.
[49] A. Abhashkumar, A. Gember-Jacobson, and Akella. (2020). *Tiramisu Source Code*. [Online]. Available: https://github.com/anubhavnidhi/batfish/tree/tiramisu
[50] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev. (2020). *Config2Spec Source Code*. [Online]. Available: https://github.com/nsg-ethz/config2spec

Ning Kang received the B.E. degree in computer science from Chang'an University in 2021. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. His research interests include network intent mining and verification.

Peng Zhang (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University in 2013. He was a Visiting Researcher with The Chinese University of Hong Kong and Yale University. He is currently a Professor with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. He is also with the MOE Key Laboratory for Intelligent Networks and Network Security. His research interests include verification and measurement.

Hao Li (Member, IEEE) received the Ph.D. degree in computer science from Xi'an Jiaotong University in 2016. He is currently an Associate Professor with the School of Computer Science and Technology, Xi'an Jiaotong University. He is also with the MOE Key Laboratory for Intelligent Networks and Network Security. His research interests include programmable networks and network functions.

Sisi Wen received the B.E. degree from Taiyuan University of Technology in 2020 and the M.S. degree from Xi'an Jiaotong University, China, in 2023. He joined the Network Engineering Team at ByteDance in 2023.

Chaoyang Ji received the master's degree from Dalian University of Technology. He is currently with the Cloud Computing and Networking Innovation Laboratory, Huawei Cloud Computing Technology Company Ltd. His research interests include cloud networking and large-scale AI training and inference.

Yongqiang Yang received the master's degree from Beihang University. He is currently the Head of the Cloud Computing and Networking Innovation Laboratory, Huawei Cloud Computing Technology Company Ltd. His research interests include cloud networking and large-scale AI training and inference.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on January 13,2026 at 03:15:03 UTC from IEEE Xplore. Restrictions apply.