Fast SMT-Based Fault Tolerance Verification for Wide Area Networks

Paper ID: #240, Number of Pages: 15+10,
Ning Kang[1], Peng Zhang[2], Hao Li[3], and Jianyuan Zhang[4]

Xi'an Jiaotong University, Xi'an, China
{[1]kangning@stu., [2]p-zhang@, [3]hao.li@, [4]2244313675@stu.}xjtu.edu.cn

Abstract. Configurations of routing protocols in wide area networks (WANs) are highly sophisticated and prone to bugs, leading to severe network outages and security breaches. SMT-based network verification can assist operators in checking the configurations, but it still faces scalability challenges when reasoning about failures: to check whether a property holds when no more than k links fail, a verifier needs to explore a tremendous space of failure scenarios. To this end, this paper proposes *VeriBoost*, a method that can leverage the topology features of WANs to reduce the space of failure scenarios, thereby improving the scalability of SMT-based verification on WANs. *VeriBoost* achieves the reduction by pruning links that are irrelevant to a property, and compressing multiple links whose failures have an equivalent impact on the property. Experiments on real WAN topologies show that it speeds up SMT-based verification by 2–47×.

Keywords: SMT · Verification · Wide Area Network · Failures.

1 Introduction

Configuring a modern network is difficult and prone to bugs, which can lead to severe network outages and security breaches [1,4,7,23,27,33,36]. These bugs are often latent: misconfigurations are hidden in tens of thousands of configuration lines, and a property violation, e.g., blackhole, is only triggered when some links or nodes fail [4]. To prevent this, SMT-based verifiers [3–5,9,24,30,32] are proposed. They can prove or disprove whether a given property holds when no more than k links or nodes fail simultaneously.

However, scalability is still one of the most important challenges faced by SMT-based verifiers, due to their implicit enumerative manner. To check whether a property holds when no more than k links fail, verifiers need to consider all $\sum_{i=0}^{k} C_l^i$ possible link failure combinations, where l is the number of links. Here, a property holds for failure tolerance k if it is satisfied under any combination of up to k link failures. Such a large SMT solving space significantly limits the scalability of verifiers. Our experiments show that to verify 300 properties with $k = 1, 2, 3$ on a 158-node WAN, SMT-based verifiers take 12 hours (see §5.3).

Some previous efforts tried to enhance the scalability of SMT-based verifiers. However, we observe they achieve limited improvement on WANs. For example, Bonsai [32] and Origami [17] exploit the symmetric structure of data center networks (DCNs) to compress the network state, whereas *WAN topologies lack such symmetry*. BiNode [26] assumes that the network configurations follow the Gao-Rexford principle [16], and Trailblazer [21] only considers the failures of links on packet forwarding paths. These methods are *limited by the complex routing policies of WANs*. Moreover, NetSMT [12] is ineffective when the property holds, i.e., when the SMT formula has no solution.

To this end, we ask "Is it possible to reduce the failure space without assuming symmetric topology structures or special routing policies?" Our answer is affirmative based on the following two observations.

1. *Failures of some links are <u>irrelevant</u> to the property of interest.* For example, a link that appears only in loop paths can never be used for the best routes, because there always exists a non-loop (i.e., *simple*) path [35] with strictly higher preference. Generally, we prove that links that do not appear on any simple path are *irrelevant* to the property and can be pruned (Theorem 1).
2. *Failures of some links have an <u>equivalent</u> impact on the property of interest.* For example, if the best route initially propagates along $path_1$, and the failure of either of two links A and B causes it to switch to $path_2$, then these two link failures have the same impact on route propagation of the property. Generally, we show that such an equivalence of links is transitive, and a group of *equivalent* links can be compressed (Theorem 3).

When realizing the SMT solving space reduction approach, we are faced with the problem: *how to efficiently identify <u>irrelevant</u> or <u>equivalent</u> links, without loss of accuracy?* First, to identify irrelevant links, a straightforward way is to find all *simple paths*, and exclude all links that appear on any of those *simple paths*. However, finding all *simple paths* on a graph is shown to be NP-hard [35]. For example, on a WAN topology containing 158 routers (189 links), computing *simple paths* for a pair of nodes using depth-first search takes more than 4 hours. Second, to identify equivalent links, we have to analyze the impact of each single link failure, and check whether any pair of links have the same impact. This requires extra verification (by solving an SMT formula), clearly an overkill.

In this paper, we introduce *VeriBoost*, a method that can efficiently reduce the failure space for SMT solvers, and thereby improve the scalability of network verification on WANs. First, *VeriBoost* leverages *point biconnected components* [31] to transform the topology into a concise tree representation (termed *block-cut tree* or *block-point tree* [18]), where each node represents a set of nodes forming a *point biconnected component*. We show that by traversing the tree we can collect all links on all *simple paths* in polynomial time. Second, instead of identifying all equivalent links which requires verification, *VeriBoost* tries to find a subset of equivalent links, solely based on graph features. For example, we show that if a node has a degree of 2, then failing the adjacent links will have the same impact on routing propagation. Based on this fact, *VeriBoost* can already identify many links that are equivalent based on our datasets. Specifically, both methods achieve a 78% reduction in the number of links (see §5.1), and their computation time is within 700μs for the real WAN topology (see Appendix A).

In summary, our contribution is three-fold:

- We propose *VeriBoost*, a new method that improves the scalability of SMT-based verifiers on WANs, by reducing the SMT solving space.
- We formally prove that the SMT solving space reduction preserves verification correctness.
- We apply *VeriBoost* to SMT-based verification method [8] and show that it achieves a speedup of 2–47× on real WAN topologies, while the state-of-the-art method [12] achieves only a 2–3× speedup. *The source code of VeriBoost is currently available (see §6) and will be open-sourced upon acceptance.*

2 Motivation

2.1 Background

Network misconfigurations. Network configurations are complex and prone to bugs. Worse still, many bugs in network configurations are latent, and can only be triggered when some nodes or links fail. For example, consider a network where a packet is normally forwarded along $path_1$. If a link on $path_1$ fails, the packet is rerouted to $path_2$. However, if a node along $path_2$ filters traffic and does not permit this packet, it will be dropped, creating a blackhole [4]. To prevent these latent bugs, researchers have developed many SMT-based verifiers that can comprehensively reason about the impact of arbitrary node/link failures. Specifically, they check whether a given property always holds when no more than k nodes/links fail. We refer to such verification as "fault tolerance" or "k-failure" verification, and use *failure scenario* to denote which links are down and which are up (see Definition 1).

SMT-based fault tolerance verification methods model the verification problem as a set of SMT formulas $N \wedge \neg P$, where N encodes the behaviors of the network under a given configuration, and P encodes the property that the network should satisfy. [3–5, 24, 30, 32]. To reason about link failures, the SMT formula becomes $N \wedge \neg P \wedge K$, with K encoding the arbitrary $\leq k$ link failures: $\sum_{l \in L} failed_l \leq k$, where $l \in L$ denotes a link, and $failed_l$ is a pseudo-boolean variable representing whether link l fails or not, with $\sum_{l \in L} failed_l$ representing the numerical sum of the variables $failed_l$, and L is the set of all links in the network. Then, the verifiers try to find a solution satisfying the SMT constraints with off-the-shelf solvers like Z3 [9]. If there is such a solution, it indicates that the property is violated under some link failures; otherwise, the property holds when no more than k links fail. Although SMT-based verifiers do not explicitly enumerate the *failure scenarios*, they still involve $O(\sum_{i=0}^{k} C_{|L|}^{i})$ iterations.

2.2 Related Work

SMT-based verifiers offer flexibility for verifying a wide range of properties, but their scalability is limited, motivating many studies to accelerate them [5, 12, 17, 21, 26]. However, these methods are ineffective for WANs, as summarized below.

Reducing encoded variables via topology symmetry. Data centers use topologies like multi-stage Clos networks [2], which are regular: symmetric with multiple layers (core, spine, and leaf), and multiple nodes have primary-backup relationships. Based on this regularity, some tools compress the large topology into a smaller one (with many fewer nodes and links) for speedup. For example, Origami [17] and Bonsai [5] compress a primary node and its backup nodes into a single one, since the routing behavior of primary and backup nodes are quite similar. However, such regularity does not hold for WANs. For example, the real WAN topologies from Topology Zoo [20] have no clear central devices or hierarchical structures, because the deployment of devices and links is closely related to geographic locations that are highly random.

Adding constraints via simplified routing policy assumptions. Complex routing policies are widely used to control route WANs connecting thousands of Autonomous Systems (ASes) operated by different institutions, such as Internet Service Providers, companies, and universities, through the complex policies of the Border Gateway Protocol (BGP) protocol [16]. Some methods tried to improve the scalability, with simplified assumptions on the routing policies, making them ineffective or raising false positives. For example, BiNode [26] assumes the network follows the Gao-Rexford principle (e.g., an AS prefers routes from customers over those from peers and providers) [16], so as to constrain the search space. However, it is hard to ensure Gao-Rexford principles on WANs, without verification. Trailblazer [21] only considers the failures of links appearing on packet forwarding paths. This implicitly assumes that packets and routes for the same prefix traverse the same path, which does not hold on WANs [19].

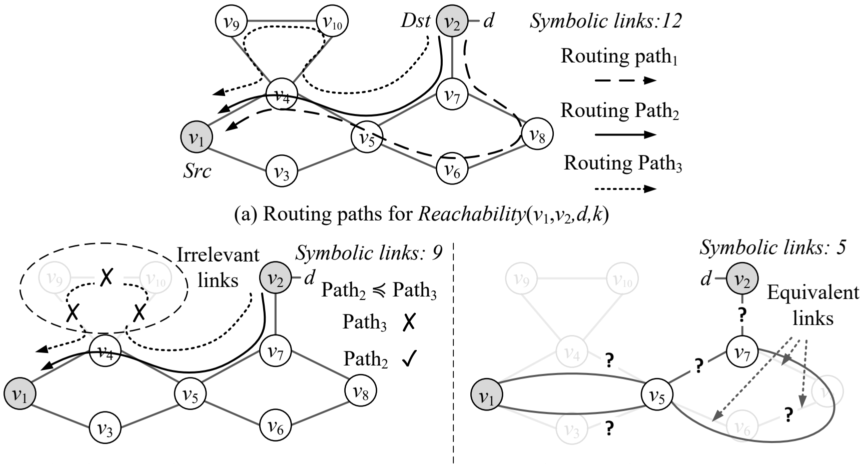
Guiding the solving process through variable ordering. NetSMT [12] guides the order of SMT solving to quickly find a solution, i.e., a counterexample that violates a property. However, when the property holds, i.e., the SMT problem does not have a solution, the SMT solver still has to explore the whole SMT solving space, making NetSMT ineffective.

Some other methods accelerate SMT-based verification by sacrificing the support for fault tolerance, and are thus not applicable to our target problem. For instance, the modular approaches LightYear [30], Timepiece [3], and Kirigami [32] require manual network partitioning and are thus applicable only to a single *failure scenario*. Moreover, ACORN [24] ignores path information in its encoding and cannot verify properties such as waypointing (i.e., ensuring that a path passes through a specific node), also restricting it to a single *failure scenario*.

3 Overview

3.1 Basic Idea

To speedup fault tolerance verification, our basic idea is to reduce the uncertainty of link state prior to SMT solving, thereby shrinking the failure space for SMT solvers. Specifically, for a given property to be verified, suppose that the state of some links are already known, including those that are “down” ($L_{\mathfrak{D}}$) and those



(a) Routing paths for *Reachability*(v_1,v_2,d,k)

(b) The network after *Link Pruning*

(c) The network after *Link Compression*

Fig. 1. An example of a BGP network with progressively applied *Link Pruning* and *Link Compression*. Each router has a different AS number and runs BGP to exchange routes. The property to be verified is reachability, that packets from source v_1 can reach destination d on v_2, even if no more than k links fail. Note that the routing path and forwarding path are in opposite directions. Our methods can be applied to any k, hence no specific tolerance is specified here.

that are "up" ($L_\mathcal{U}$). We then treat the remaining links as symbolic links $L_\mathcal{S} = L \setminus (L_\mathcal{U} \cup L_\mathcal{D})$, whose status is unknown (either up or down). This allows SMT-based verifiers to encode only the links in $L_\mathcal{S}$ as boolean variables, rather than all links, significantly reducing the solving space. Recall that SMT-based verifiers need to check $N \wedge \neg P \wedge K$, where $K \triangleq \sum_{l \in L} failed_l \leq k$. *VeriBoost* transforms K into the following constraint, enabling more efficient verification.

$$K \triangleq \bigwedge_{l \in L_\mathcal{U}} failed_l = 0 \bigwedge_{l \in L_\mathcal{D}} failed_l = 1 \wedge \sum_{l \in L_\mathcal{S}} failed_l \leq k \tag{1}$$

To this end, we present two observations that allow us to identify these "down" and "up" links without assuming any regular topology structure or specific routing policy, enabling applicability to WANs.

Observation 1: For a given property, many failure scenarios are irrelevant. To illustrate this, we consider the example in Fig. 1 (a), where each router has a different AS number and runs BGP to select the best routes. According to the best route selection of BGP [13, 25], a router will prefer a route with higher local preference and then the AS path length. As specified in [13,25], the local preference is not propagated among different ASes. Now, the best route is determined by the AS path length (with a shorter path being preferred). Then, routes propagated along $path_2$ (the AS path length is 4) are always preferred by node v_1, over the routes propagated along $path_3$ (the AS path length is 7), since $path_2$ is a subsequence of $path_3$. Therefore, the links on $path_3$ (v_4-v_9) will never be used by any best route for the property.

Based on this observation, when verifying *reachability*(v_1, v_2, d, k), i.e., packets sent from node v_1 can reach prefix d[1] on node v_2 even failing any k links, we can first check whether there are policies that decrease the AS path length (which is quite uncommon), and if not, we can safely prune the links. We summarize that for a given property, a link excluded by all *simple paths*[2] [35] will not propagate the best route. Thus, we compute the set of all links included in *simple paths* and set the complement of these links as "down" status for the verifiers. For example, the three links v_4-v_9, v_4-v_{10}, and v_9-v_{10} are reduced in Fig. 1(b). In this example, the remaining symbolic links amount to $12 - 3 = 9$ links. This effect is more significant in real WAN topology, where the number of links is reduced from 188 to 126 (a $(188 - 126)/188 = 33\%$ reduction) [20], detailed in Appendix B).

Observation 2: For a given property, many failure scenarios have the same equivalent impact. To show this, we consider the example after *Link Pruning* in Fig. 1(b), where routes at node v_8 propagate only in two directions, i.e., to v_6 or v_7. Suppose the best route from v_2 to v_1 is propagated via $path_1$ instead of $path_2$ due to smaller route metric values. When the link v_7-v_8 or v_8-v_6 fails, the best routing path changes from $path_1$ to $path_2$ because the failure of either v_7-v_8 or v_8-v_6 blocks v_5 from selecting the route via v_6. Thus, failing v_7-v_8 and v_8-v_6 individually or simultaneously have the same impact on route propagation for the property, and we refer to them as *equivalent links*.

Based on this observation, when verifying *reachability*(v_1, v_2, d, k), we can remove one of the variables for those links, v_7-v_8 or v_8-v_6, to create a smaller model. We summarize that, for each set of equivalent links, we randomly select one link as a potentially failed link. The remaining links within this set, equivalent to the potentially failed link, are considered "up" status for the verifiers. For example, four links v_1-v_4, v_1-v_3, v_5-v_6, and v_7-v_8 are reduced in Fig. 1(c). In this example, the remaining symbolic links are $(9 - 4) = 5$ links. This effect is significant in real WAN topology, where the number of links is reduced from 126 to 61 (a $(126 - 61)/188 = 34\%$ reduction), detailed in Appendix B).

We highlight that the above methods make no assumptions about topology structures or routing behaviors that are hard to guarantee. For example, to ensure that $path_2$ is always preferred over $path_3$, we only need to confirm "the preference of a path does not decrease when prefixed by a link." This, in turn, assumes that "BGP does not propagate local preference across AS," as standardized by [13, 25], and that "AS path length never decreases," which can be easily pre-checked. If some routers on the path are configured with routing policies that modify the AS path (which is quite uncommon [4, 22, 28]), the reduction can be disabled to avoid false positives.

[1] The prefix d (i.e., a block of IP addresses, e.g., 112.0.0.1-112.0.0.255) is connected to node v_2. Although the parameter v_2 can be omitted, we retain it for clarity.

[2] Some literature [10, 34] uses the terms "walk" and "path" to distinguish between sequences that contain repeated vertices and those that do not, respectively. Other literature [35] uses "path" and "simple path" for the same distinction. For clarity, we adopt the latter terminology, e.g., using "routing path" instead of "routing walk."

3.2 Challenges

Challenge 1: How to calculate the links that never appear on any *simple path* without enumerating all *simple paths*? From Observation 1, we know that links that never appear on any *simple path* are irrelevant links (Theorem 1). To identify these links, an intuitive approach would be to first find all *simple paths* and then compute the set of links not contained in these paths. However, identifying all *simple paths* is shown to be NP-hard [35], as it requires enumerating all paths between two nodes. Moreover, on the UScarrier dataset, computing all *simple paths* between a single pair of nodes took over 4 hours (arbitrarily selecting 10 pairs of far-apart nodes), whereas verifying a property for $k = 3$ required only 200 seconds (see §5.3).

Challenge 2: How to judge if two failed links have an equivalent impact on the best routes without calculating the best routes? From Observation 2, we know that two links whose failures have an equivalent impact can be compressed. An intuitive approach to achieve this relies on calculating the best routes. However, implicitly or explicitly computing the best routes would involve performing verification again, creating a chicken-and-egg problem, but we aim to identify equivalent impacts before verification.

4 Details

In this section, we first provide the definitions in §4.1. We then introduce *Link Pruning* in §4.2 and *Link Compression* in §4.3 (with formal proof) to address the challenges described in §3.2. Finally, we show the optimization.

4.1 Network Model

Property. Let $T = (V, L)$ be a network topology, where V is the set of vertices and L is the set of links. Let $property(v_1, v_2, d, k)$ be the property that packets from source node v_1 to prefix d on node v_2 even if any k links fail, where $v_1 \in V$ is the source (destination) node, and $v_2 \in V$ is the destination (source) node in forwarding (routing). Here, representative properties include [6, 19, 21, 29, 37]:

- *Reachability*(s,n,d,k): packets sent from s can reach d on n.
- *Waypoint*(s,n,w,d,k): packets sent from s to d on n, passing through w.
- *Loadbalance*(s,n,m,d,k): packets sent from s to d on n, along m paths.

Failure scenario. We now clarify the link states. Before verification, links have three states: down, up, and symbolic (unknown). During verification, the verifiers will assign symbolic links a concrete state: either up or failed.

Definition 1. *We define two types of failure scenarios.*

(i) A concrete failure scenario $f = (L_{\mathfrak{U}}, L_{\mathfrak{D}})$ *partitions links* L *into two subsets:* $L_{\mathfrak{U}}$ *(up) and* $L_{\mathfrak{D}}$ *(down), i.e., all links have concrete states.*

(ii) *A symbolic failure scenario* $F = (L_{\mathcal{U}}, L_{\mathcal{D}}, L_{\mathcal{S}}, k)$ *partitions links* L *into three subsets:* $L_{\mathcal{U}}$ *(up),* $L_{\mathcal{D}}$ *(down), and* $L_{\mathcal{S}}$ *(symbolic). Here,* k *is the number of links in* $L_{\mathcal{S}}$ *that can fail simultaneously.*

(iii) *We say a symbolic failure scenario* F *contains a concrete failure scenario* f *or* $f \in F$*, if* $F.L_{\mathcal{U}} \subseteq f.L_{\mathcal{U}}$*,* $F.L_{\mathcal{D}} \subseteq f.L_{\mathcal{D}}$*, and* $|f.L_{\mathcal{D}} \backslash F.L_{\mathcal{D}}| \leq k$*, where* $\backslash$ *represents the set difference operation.*

The third definition states that a *symbolic failure scenario* contains all *concrete failure scenarios* satisfying the links in $F.L_{\mathcal{U}}$ are up, the links in $F.L_{\mathcal{D}}$ are down, and no more than k links are failed in $F.L_{\mathcal{S}}$. Taking Fig. 1(c) as an example, a *symbolic failure scenario* $(L_{\mathcal{U}}, L_{\mathcal{D}}, L_{\mathcal{S}}, 2)$, with $L_{\mathcal{S}} = \{v_4\text{-}v_5, v_3\text{-}v_5, v_5\text{-}v_7, v_6\text{-}v_8, v_2\text{-}v_7\}$, contains $C_5^2 = 10$ *concrete failure scenarios.*

Based on the notion of *symbolic failure scenario*, most existing network verifiers [4, 15] handle a *symbolic failure scenario* of $(\varnothing, \varnothing, L, k)$; while we check $(L_{\mathcal{U}}, L_{\mathcal{D}}, L_{\mathcal{S}}, k)$, and reduce the size of $L_{\mathcal{S}}$ as much as possible before solving.

Related routes. Let $\mathcal{R}_f(v_1, v_2, d)$ denote the best routes of the devices among the forwarding and routing paths for the prefix d of *property*(v_1, v_2, d, k) under *concrete failure scenario* f (The rigorous definition is provided in Appendix C).

4.2 Link Pruning

Observation 1 states that for the given property, the links that do not appear on any *simple path* are irrelevant and can be pruned. In the following, we formalize this observation with a theorem and then prove the theorem.

First, we define irrelevant links and *simple paths*.

Definition 2. *Given a network topology* $T = (V, L)$*, and property*(v_1, v_2, d, k) *to verify, a link* l *is said to be "irrelevant", if the best routes related to the prefix* d *are the same whether* l *fails or not, on any concrete failure scenario. Formally, this can be expressed as:*

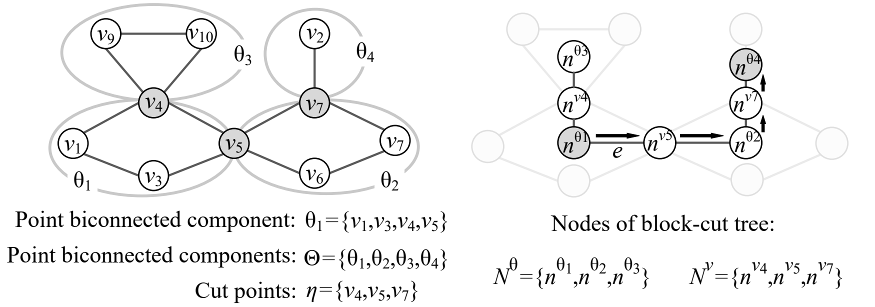
$$\begin{aligned} \textit{for } \forall f = (L_{\mathcal{U}}, L_{\mathcal{D}}) : \mathcal{R}_{(L_{\mathcal{U}} \backslash \{l\}, L_{\mathcal{D}} \cup \{l\})}(v_1, v_2, d) = \\ \mathcal{R}_{(L_{\mathcal{U}} \cup \{l\}, L_{\mathcal{D}} \backslash \{l\})}(v_1, v_2, d) \end{aligned} \tag{2}$$

Next, we describe how to identify these irrelevant links.

Definition 3. *For a graph* $T = (V, L)$*, a path (or walk) from* v_0 *to* v_k *(of length* k*) is a non-empty alternating sequence* $v_0, l_1, v_1, l_2, v_2, \ldots, l_k, v_k$ *of vertices and links in* T *such that* $l_i = (v_i, v_{i+1})$ *for all* $i < k$ [10].

Definition 4. *For a graph* $T = (V, L)$*, a path from* $u \in V$ *to* $v \in V$ *is said to be a simple path of* T *if the vertices in a path are all distinct* [10].

Theorem 1. *For a network topology* $T = (V, L)$ *and property*(v_1, v_2, d, k)*, if a link does not appear on any simple path of* T*, then the link is irrelevant.*


v_9
v_{10}
θ_3
v_2
θ_4
v_4
v_7
v_1
v_5
v_7
θ_1
v_3
v_6
θ_2
Point biconnected component: $\theta_1=\{v_1,v_3,v_4,v_5\}$
Point biconnected components: $\Theta=\{\theta_1,\theta_2,\theta_3,\theta_4\}$
Cut points: $\eta=\{v_4,v_5,v_7\}$
n^{θ_3}
n^{v_4}
n^{θ_1}
e
n^{v_5}
n^{θ_4}
n^{v_7}
n^{θ_2}
Nodes of block-cut tree:
$N^{\theta}=\{n^{\theta_1},n^{\theta_2},n^{\theta_3}\}$
$N^{v}=\{n^{v_4},n^{v_5},n^{v_7}\}$

(a) The illustration of basic conception (b) Block-cut tree for the example topology

Fig. 2. The example of the *block-cut tree* for the network in Fig. 1.

To prove this theorem, we make two assumptions: (1) BGP converges to a unique state, which is true in most practical cases [14, 29]. (2) The network exhibits monotonicity, meaning that the weight of a path does not decrease when prefixed by a link. This generally holds in common routing protocols, e.g., BGP and open shortest path first (OSPF) [4, 22, 28]. The formal description of these two assumptions can be found in Appendix C.

Proof. (proof sketch) For clarity, we only provide a proof sketch here, and the full proof is in Appendix D. To prove that a link is irrelevant, according to Definition 2, we only need to prove that the best routes among the forwarding path and routing path before and after failing this link remain the same. We prove this by induction, working on the number of routing propagation hops[3].

Then, the problem is how to find all links that do not appear on any *simple paths*. Note that the straightforward way of computing all *simple paths* is NP-Hard [35]. To efficiently identify all irrelevant links, we present a polynomial algorithm. Our algorithm is heavily based on *point biconnected component* (PBC), a concept in graph theory [31]. In the following, we introduce the definition of PBC.

Definition 5. *Given a graph T, and a pair of nodes (u, v), if the deletion of any single node in T (except u and v) does not disconnect u and v, then we say that u and v are biconnected. If all pairs of two nodes in a graph are biconnected, then we say the graph is point biconnected. If a subgraph H of T is maximal point biconnected, meaning it is a point biconnected subgraph of T and no point biconnected subgraph of T contains H as a proper subgraph, then we say that H is a point biconnected component.*

We will use θ_i to represent a PBC, and use Θ to represent the set of all PBCs of the graph. For the example in Fig. 2, one PBC is $\theta_1 = \{v_1, v_3, v_4, v_5\}$, and $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$.

Definition 6. *A graph is connected if there exists a path between every pair of distinct vertices. A cut point in a connected graph $G = (V, E)$ is a vertex $v \in V$ such that the deletion of v causes the graph to become disconnected.*

[3] We omit the proof for forwarding paths, as it is quite similar.

Algorithm 1: Links Pruning

```
   input  : $T = (V, L)$: the network topology, $v_1$: the source node of the
            property, $v_2$: the destination node of the property.
   output : $L_{irr}$: the set of irrelevant links in $L$.
1  Function IrrelevantLinks($v_1, v_2, T$):
2      $\eta, \Theta \leftarrow$ Tarjan($T$)                            ▷ computing PBC
3      $\mathcal{T} \leftarrow$BCTree($\eta, \Theta$)                         ▷ constructing $\mathcal{T}$
4      $n_1 \leftarrow$ Node2PBC ($v_1, \mathcal{T}$)                       ▷ src node in $\mathcal{T}$
5      $n_2 \leftarrow$ Node2PBC ($v_2, \mathcal{T}$)                       ▷ dst node in $\mathcal{T}$
6      $N' \leftarrow$ Traverse($n_1, n_2, \mathcal{T}$)                  ▷ finding path on $\mathcal{T}$
7      $\Theta' \leftarrow$ Mono($\{\theta | n^\theta \in N'\}$)                ▷ checking configs
8      $V' \leftarrow \{v \in \theta | \theta \in \Theta'\}$
9      $L' \leftarrow \{(v_i, v_j) | v_i \in V', v_j \in V', (v_i, v_j) \in L\}$
10     return $L \backslash L'$                              ▷ returning irrelevant links
```

We will use η to represent the set of all *cut points* of a graph. For the example in Fig. 2, $\eta = \{v_4, v_5, v_7\}$.

There is a useful property that a cut point is included in two different point biconnected components [31]. Based on it, we define the *block-cut tree* (or *block-point tree*) [18].

Definition 7. *For an undirected graph $T = (V, L)$, a graph $\mathcal{T}$ is the block-cutpoint tree of T if it satisfies : (1) (Block vertices) For each biconnected component θ of T, there is a corresponding vertex n^θ in $\mathcal{T}$. (2) (Cut vertices) For each cut point v of T, there is a corresponding vertex n^v in $\mathcal{T}$. (3) An edge (n^v, n^θ) exists in $\mathcal{T}$ iff v is a vertex in θ.*

According to [18], $\mathcal{T}$ is shown to be a tree. For example, Fig. 2 (b) shows the *block-cut tree* of the network topology T in Fig. 2 (a).

Alg. 1 summarizes the process to identify all the irrelevant links in a network.

Step1. Computing point biconnected components. First, for a given topology, we obtain *cut points* η and *point biconnected components* (PBCs) Θ using Tarjan's algorithm [31] (Line 2).

Step2. Constructing block-cut tree. Next, we construct the block-cut tree (Line 3). Specifically, given a network topology T with *point biconnected component* set Θ and cut point set η, its block-cut tree BCTree(η, Θ) is a graph $\mathcal{T} = (\left\{N^v \cup N^\theta\right\}, E)$, where:

$$N^\theta = \left\{n^\theta | \theta \in \Theta\right\}, N^v = \left\{n^v | v \in \eta\right\} \tag{3}$$

$$E = \left\{e | e = (n^v, n^\theta) \vee (n^\theta, n^v), v \in \theta\right\} \tag{4}$$

Next, we find nodes n_1 and n_2 in $\mathcal{T}$ that correspond to the PBCs of the source node v_1 and destination node v_2 with Node2PBC (Lines 4-5).

Step3. Identifying irrelevant links. We compute all PBCs on path from n_1 to n_2 on $\mathcal{T}$, with Traverse (Line 6). Note that the source node and destination node may not be within the same PBC. For example, from v_2 to v_1, it may need to pass through three PBCs $\{\theta_1, \theta_2, \theta_4\}$. To confirm monotonicity, we use a function Mono to check that all nodes in a PBC are configured either with pure BGP, where the AS path length never decreases, or with OSPF. If not, we remove all links of the PBC from the down links (Line 7). We then collect all links L' within these PBCs (Lines 8-9). Finally, we return the complement set of L' as irrelevant links (Line 10).

Time complexity. The complexity of IrrelevantLinks is $O(|V| + |L|)$, because calculating the *point biconnected components* takes $O(|V| + |L|)$ [31], and the remaining part has a lower time complexity. Specifically, we could mark nodes that are configured with pure BGP and without modifications to the AS path during configuration parsing, and then Mono requires $O(|N|)$ in this stage; Since $|E| < |L|$, Traverse traverses each edge in E once, which takes less than $O(|L|)$; Since each link will be added to L' at most once, this requires $O(|L|)$.

Proof of correctness. We prove that Alg. 1 can find all irrelevant links with the Theorem 2.

Theorem 2. *For $T = (V, L)$ and property(v_1, v_2, d, k), if $l \in$ IrrelevantLinks (v_1, v_2, T) in Alg. 1, then l does not appear on any simple path from v_1 to v_2.*

The full proof is in Appendix E.

4.3 Link Compression

Observation 2 states that if the failures of two links have the same impact on the propagation of routes, then these two links are equivalent and can be compressed. We formally define our notion of equivalence as follows.

Definition 8. *Given a network topology $T = (V, L)$, and property(v_1, v_2, d, k) to verify, links l_1 and l_2 are said to be "equivalent" if the best routes related to prefix d are the same either l_1 or l_2 fails, on any concrete failure scenario, formally stated as:*

$$\begin{aligned} \textit{for } \forall f = (L_{\mathcal{U}}, L_{\mathcal{D}}) : \mathcal{R}_{(L_{\mathcal{U}} \setminus \{l_1\}, L_{\mathcal{D}} \cup \{l_1\})}(v_1, v_2, d) = \\ \mathcal{R}_{(L_{\mathcal{U}} \setminus \{l_2\}, L_{\mathcal{D}} \cup \{l_2\})}(v_1, v_2, d) \end{aligned} \tag{5}$$

By compressing equivalent links into a single one, we can reduce the SMT solving space. Then, the problem is how to identify the equivalent links? The straightforward way is to fail each of the two links and analyze their impact of these failures, which is equivalent to another round of verification (solving SMT problem or simulating route computation). This clearly is an overkill for this problem. We set out for a less ambitious goal: instead of identifying all equivalent links, we try to find a subset of equivalent links solely based on the graph features. Next, we show the algorithm, and prove its correctness.

Definition 9. *For $T = (V, L)$, a path from $u \in V$ to $v \in V$ is said to be a "trivial path" of T if the degrees of u and v are not 2, and other nodes (except u and v) have degree 2.*

Theorem 3. *For $T = (V, L)$ and property(v_1, v_2, d, k), if two links l_1 and l_2 do not both contain either v_1 or v_2 and appear on a trivial path of T, then l_1 and l_2 are equivalent.*

Proof. (proof sketch) The full proof is in Appendix F. To prove that two links are equivalent, we need to show that the best routes among the forwarding path and routing path after failing these two links remain the same. The proof is similar to the proof of Theorem 1, both using induction. The difference is that the induction is based on the number of endpoints of trivial paths.

Actually, a group of equivalence links has the property that any two links within the group are equivalent, which allows us to compress all the links within the group into a single link. This is due to the transitivity of equality in Definition 8, the equivalence of links also exhibits transitivity: if l_1 and l_2 are equivalent, and l_1 and l_3 are equivalent, then l_2 and l_3 are also equivalent. Due to this transitivity, we know all links in one trivial path are all equivalent.

For a given property, we use a depth-first search to traverse all links and nodes once. This allows us to identify all *trivial paths*. Next, for each *trivial path*, the links on it form a link set S. Then, we randomly select one link as a symbolic link from S, which may potentially fail. The remaining links within S are all treated as equivalent links to this symbolic link, and are treated as up links. We call this method a function EquivalentLinks(v_1, v_2, T).

Time complexity. The complexity of EquivalentLinks is $O(|V| + |L|)$, as it traverses all links and nodes twice, i.e., find all trivial paths and symbolic links.

Proof of correctness. We prove the correctness of EquivalentLinks following Theorem 3.

4.4 Optimizations

Property trimming. *VeriBoost* pre-checks the connectivity of the property on the network topology before *Link Pruning* and *Link Compression*. If the tolerance value of a property to be verified is k, there must be more than k disjoint paths in the topology. Here, we use the minimum cut to effectively calculate the number of disjoint paths. If the property fails this check, *VeriBoost* promptly returns it as violated.

5 Evaluation

We evaluate *VeriBoost* on real topologies to address the following questions:

- Is *VeriBoost* effective on different irregular topologies? We show that *VeriBoost* achieves a 40% reduction in symbolic links across all 260 real topologies, and an 80% reduction in 78% of them (§5.1).
- Is *Veriboost* correct on real datasets? We show that our verification results are identical to the results of existing verifiers (§5.2).

Table 1. The statistics of WAN datasets, where datasets with "-O" are OSPF datasets, and others are BGP datasets.

Datasets	Ren	Arn	Bic-O	Bic	Esn	Lat	Col-O	Col	Clt	Usc-O	Usc	Cog
Category	Small				Medium				Large			
Nodes	34	35	33	33	69	70	70	70	154	158	158	198
Links	44	47	48	48	80	75	85	85	178	189	189	244
Lines ($\times 10^3$)	4.8	5.0	3.0	6.9	9.2	9.1	6.1	11.5	20.6	13.7	22.0	27.1

- How does *VeriBoost* speed up SMT-based verifier? We show that it speeds up the SMT-based verifier by 2-47× (§5.3).
- How do optimizations contribute to *VeriBoost*? We show that *Link Pruning*, *Link Compression* contribute similarly, and *Property Trimming* contributes little; The time for them ranges from microseconds to milliseconds, which is negligible, compared to the seconds-level verification. (For clarity, we present these two experiments in Appendix A and report only the conclusions here.)

Implementation. We implemented *VeriBoost* with 4k lines of Java code, and applied it to the SMT-based verifier.

Approaches for comparison. We choose Minesweeper [4] as the SMT-based baseline for comparison, because it is one of the most feature-complete SMT-based verifiers and has been widely adopted as a baseline by many works [5,12,17,21,26]. We also compare with NetSMT [12], the state-of-the-art speedup method for SMT-based verification. We do not compare to other speedup methods for the following reasons. Origami [17] and Bonsai [5] target only data center networks (DCNs) [12]. Trailblazer has false positives due to missing hot edges as analyzed in [19]. As for BiNode, we found it to be slower than Minesweeper on our WAN datasets, which involve solely the BGP protocol or OSPF protocol[4]. This adds overhead without being paid off on our WAN datasets. Other methods, such as ACORN [24], Kirigami [32], Lightyear [30], and Timepiece [3], do not support fault tolerance verification.

Properties. We consider three properties as shown in §4.1 and select 25 pairs that can possibly hold for each $k = 0, 1, 2, 3$. Thus, we have $3 \times 4 \times 25 = 300$ properties for each network. We verify 300 properties because Minesweeper times out after 12 hours on the Uscarrier dataset alone. For a fair comparison, we select pairs that avoid cases where both nodes fall within a *point biconnected component* containing only a few nodes, thereby worsening the speedup of *VeriBoost*.

Datasets. *VeriBoost* has two stages: (1) reducing the symbolic links and (2) applying the reduction to verifiers. For the first step, we use the all 260 real-world WAN network topologies from the Topology Zoo [20] for the comparison experiments. For the second stage, real-world WAN configurations are typically confidential and not publicly available. Therefore, existing works [6, 12] mostly rely on synthesizing configurations from real topologies using the approach in [11]. To avoid arbitrarily selecting unrepresentative topologies, we follow prior

[4] To handle the dependency between BGP and OSPF, BiNode replicates the variables associated with each BGP route.

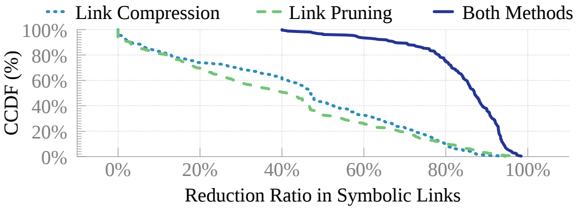


Fig. 3. The CCDF of the *reduction ratio* for 260 real topologies, when applying *Link Compression*, *Link Pruning*, or both methods.

work. As summarized in Table 1, Fang et al. [12] considered nine representative topologies for BGP configuration synthesis, while Birkner et al. [6] considered three for OSPF configuration synthesis.

Setup. All experiments run on a Linux server with two 16-core Intel Xeon Gold 6226R CPUs @2.9GHz and 256G memory.

5.1 Effectiveness of Veriboost

We evaluate the effectiveness across 260 real WAN topologies. For each topology, we use *VeriBoost* to compute the average number of symbolic links across all-pair properties. We quantify the effectiveness using the *reduction ratio*, calculated by the following equation:

$$reduction\ ratio = \frac{\#\ of\ reduced\ links}{\#\ of\ original\ links} \tag{6}$$

Fig. 3 shows that *VeriBoost* achieves a 40% *reduction ratio* in symbolic links for 100% of the topologies, while achieving a 78% *reduction ratio* for 80% of the topologies. Moreover, *Link Pruning* and *Link Compression* yield similar improvements, indicating that applying them simultaneously is necessary.

5.2 Correctness of Veriboost

We verified 300 properties with $k = 1, 2, 3$. For SMT-based verification, the properties that hold for Minesweeper [4], NetSMT [12] and *VeriBoost* are consistent. The numerical results are provided in the Appendix, as shown in Table 2.

5.3 Speedup for SMT-based Verification

We separately verified 300 properties with $k = 1, 2, 3$ respectively using Minesweeper, NetSMT, and *VeriBoost*. Fig. 4 and Fig. 6 show that *VeriBoost* accelerates Minesweeper by 2× to 47×. In contrast, NetSMT only speeds up by 2-3×. Although the speedup of *VeriBoost* varies across different datasets, overall, it increases with the network scale and the tolerance values of the properties being verified, showing that *VeriBoost* benefits more in larger search spaces.

Fig. 5 shows the details of the speedup of NetSMT and *VeriBoost* for verifying a single property (We show one dataset for each size and omit the results for

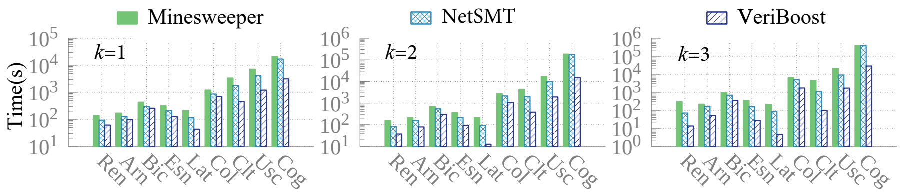


Fig. 4. Comparison of SMT-based verification for BGP datasets: Total time for verifying 300 properties, using Minesweeper, NetSMT, and *VeriBoost*.

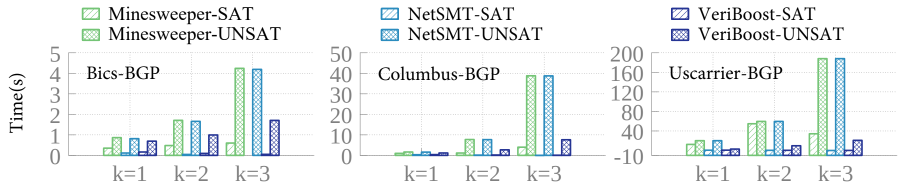


Fig. 5. Comparison of SMT-based verification: Average time for verifying a single property on BGP networks under different tolerance levels using Minesweeper, NetSMT, and *VeriBoost*. The SAT (UNSAT) time corresponds to the average time when the property is violated (holds). Negative values (Y-axis) mainly clarify the differences.

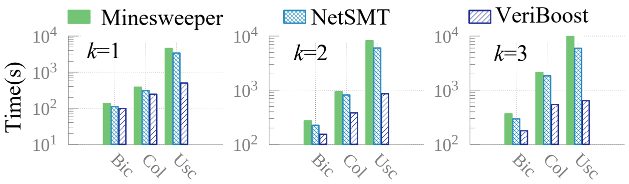


Fig. 6. Comparison of SMT-based verification for OSPF datasets: Total time for verifying 300 properties, using Minesweeper, NetSMT, and *VeriBoost*.

OSPF for clarity.). We distinguish between two types of properties: for properties that hold, there are no solutions to the SMT problem (UNSAT); for properties that do not hold, there exist one or more solutions to the SMT problem (SAT). For SAT properties, both NetSMT and *VeriBoost* achieve more than one order of magnitude speedup. For UNSAT properties, NetSMT provides no improvement, whereas *VeriBoost* still delivers roughly an order of magnitude speedup.

6 Conclusion

We propose *VeriBoost*, a tool that can accelerate SMT-based verifiers in WANs by reducing the SMT solving space before verification. Specifically, *VeriBoost* uses *Link Pruning* to prune irrelevant links and *Link Compression* to compress equivalence links in WANs. We implement *VeriBoost* in SMT-based method, and evaluate *VeriBoost* on real topologies. Results show that *VeriBoost* can scale better to large WANs and is faster than the SOTA tool by an order of magnitude. In future work, we aim to extend *VeriBoost* to accelerate simulation-based verifiers.

Data Availability. The artifact, including the source code and experiment logs, is available at https://anonymous.4open.science/r/veriboost-fm26/.

References

1. Abhashkumar, A., Gember-Jacobson, A., Akella, A.: Tiramisu: Fast multilayer network verification. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 201–219 (2020). https://doi.org/10.23919/ifipnetworking55013.2022.9829765
2. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. ACM SIGCOMM computer communication review **38**(4), 63–74 (2008). https://doi.org/10.1145/1402958.1402967
3. Alberdingk Thijm, T., Beckett, R., Gupta, A., Walker, D.: Modular control plane verification via temporal invariants. Proceedings of the ACM on Programming Languages **7**(PLDI), 50–75 (2023). https://doi.org/10.1145/3591222
4. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: A general approach to network configuration verification. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 155–168 (2017). https://doi.org/10.1145/3098822.3098834
5. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: Control plane compression. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 476–489 (2018). https://doi.org/10.1145/3230543.3230583
6. Birkner, R., Drachsler-Cohen, D., Vanbever, L., Vechev, M.: Config2spec: Mining network specifications from network configurations. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 969–984 (2020)
7. Brown, M., Fogel, A., Halperin, D., Heorhiadi, V., Mahajan, R., Millstein, T.: Lessons from the evolution of the batfish configuration analysis tool. In: Proceedings of the ACM SIGCOMM 2023 Conference. pp. 122–135 (2023). https://doi.org/10.1145/3603269.3604866
8. Brown, M., Fogel, A., Halperin, D., Heorhiadi, V., Mahajan, R., Millstein, T.: The open-source code of batfish (2023), https://github.com/batfish/batfish
9. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
10. Diestel, R.: Graph theory. Springer (print edition); Reinhard Diestel (eBooks) (2024). https://doi.org/10.1007/978-3-662-53622-3
11. El-Hassany, A., Tsankov, P., Vanbever, L., Vechev, M.: Netcomplete: Practical {Network-Wide} configuration synthesis with autocompletion. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). pp. 579–594 (2018)
12. Fang, X., Ding, F., Huang, B., Wang, Z., Han, G., Yang, R., You, L., Xiang, Q., Kong, L., Liu, Y., et al.: Network can help check itself: Accelerating smt-based network configuration verification using network domain knowledge. IEEE/International Conference on Computer (2023). https://doi.org/10.1109/infocom52122.2024.10621215
13. Feamster, N., Rexford, J.: Network-wide prediction of bgp routes. IEEE/ACM Transactions on Networking **15**(2), 253–266 (2007). https://doi.org/10.1109/tnet.2007.892876
14. Feamster, N., Rexford, J.: Network-wide prediction of bgp routes. IEEE/ACM Transactions on Networking **15**(2), 253–266 (2007). https://doi.org/10.1109/tnet.2007.892876

15. Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R., Millstein, T.: A general approach to network configuration analysis. In: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). pp. 469–483 (2015)
16. Gao, L., Rexford, J.: Stable internet routing without global coordination. IEEE/ACM Transactions on networking **9**(6), 681–692 (2001). https://doi.org/10.1145/345063.339426
17. Giannarakis, N., Beckett, R., Mahajan, R., Walker, D.: Efficient verification of network fault tolerance via counterexample-guided refinement. In: Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II 31. pp. 305–323. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_18
18. Harary, F.: Graph theory (on Demand Printing of 02787). CRC Press (2018). https://doi.org/10.1201/9780429493768
19. Kang, N., Zhang, P., Li, H., Wen, S., Ji, C., Yang, Y.: Network specification mining with high fidelity and scalability. In: 2023 IEEE 31st International Conference on Network Protocols (ICNP). pp. 1–11. IEEE (2023). https://doi.org/10.1109/icnp59255.2023.10355598
20. Knight, S., Nguyen, H.X., Falkner, N., Bowden, R., Roughan, M.: The internet topology zoo. IEEE Journal on Selected Areas in Communications **29**(9), 1765–1775 (2011). https://doi.org/10.1109/jsac.2011.111002
21. Liu, Y., Subotic, P., Letier, E., Mechtaev, S., Roychoudhury, A.: Efficient smt-based network fault tolerance verification. In: International Symposium on Formal Methods. pp. 92–100. Springer (2023). https://doi.org/10.1007/978-3-031-27481-7_7
22. Lopes, N.P., Rybalchenko, A.: Fast bgp simulation of large datacenters. In: Verification, Model Checking, and Abstract Interpretation: 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13–15, 2019, Proceedings 20. pp. 386–408. Springer (2019). https://doi.org/10.1007/978-3-030-11245-5_18
23. Moss, S.: Microsoft azure outage blamed on wan router ip change (2023), https://www.datacenterdynamics.com/en/news/microsoft-azure-outage-blamed-on-wan-router-ip-change/
24. Raghunathan, D., Beckett, R., Gupta, A., Walker, D.: Acorn: Network control plane abstraction using route nondeterminism. In: FMCAD. pp. 261–272 (2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_33
25. Rekhter, Y., Li, T., Hares, S.: Rfc 4271: A border gateway protocol 4 (bgp-4) (2006). https://doi.org/10.17487/rfc4271
26. Shao, X., Chen, Z., Holcomb, D., Gao, L.: Accelerating bgp configuration verification through reducing cycles in smt constraints. IEEE/ACM Transactions on Networking **30**(6), 2493–2504 (2022). https://doi.org/10.1109/tnet.2022.3176267
27. Sharwood, S.: Facebook rendered spineless by buggy audit code that missed catastrophic network config error (2021), https://www.theregister.com/2021/10/06/facebook_outage_explained_in_detail/
28. Sobrinho, J.L.: An algebraic theory of dynamic network routing. IEEE/ACM Transactions on Networking **13**(5), 1160–1173 (2005). https://doi.org/10.1109/tnet.2005.857111
29. Steffen, S., Gehr, T., Tsankov, P., Vanbever, L., Vechev, M.: Probabilistic verification of network configurations. In: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. pp. 750–764 (2020). https://doi.org/10.1145/3387514.3405900

30. Tang, A., Beckett, R., Benaloh, S., Jayaraman, K., Patil, T., Millstein, T., Varghese, G.: Lightyear: Using modularity to scale bgp control plane verification. In: Proceedings of the ACM SIGCOMM 2023 Conference. pp. 94–107 (2023). https://doi.org/10.1145/3603269.3604842
31. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM journal on computing **1**(2), 146–160 (1972). https://doi.org/10.1109/swat.1971.10
32. Thijm, T.A., Beckett, R., Gupta, A., Walker, D.: Kirigami, the verifiable art of network cutting. IEEE/ACM Transactions on Networking (2024). https://doi.org/10.1109/tnet.2024.3360371
33. Tom Strickx, J.H.: Cloudflare outage on june 21, 2022 (2022), https://blog.cloudflare.com/cloudflare-outage-on-june-21-2022/
34. West, D.B.: Introduction to Graph Theory, vol. 2. Prentice Hall, Upper Saddle River (2001)
35. Yang, R., Gao, R., Zhang, C.: A new algebraic approach to finding all simple paths and cycles in undirected graphs. In: 2015 IEEE International Conference on Information and Automation. pp. 1887–1892. IEEE (2015). https://doi.org/10.1109/icinfa.2015.7279596
36. Zhang, P., Gember-Jacobson, A., Zuo, Y., Huang, Y., Liu, X., Li, H.: Differential network analysis. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 601–615 (2022)
37. Zhang, P., Wang, D., Gember-Jacobson, A.: Symbolic router execution. In: Proceedings of the ACM SIGCOMM 2022 Conference. pp. 336–349 (2022). https://doi.org/10.1145/3544216.3544264

A Microbenchmark

Impact of each technique. In this part, we study the contribution of the three techniques of *VeriBoost*, *Link Pruning*, *Link Compression* and *Property Trimming*. Fig. 7 shows the speedup of total verification as *Link Pruning*, *Link Compression* and *Property Trimming* are progressively applied in SMT-based method. It is evident that both *Link Pruning* and *Link Compression* contribute to pruning effects that become more significant as the tolerance increases.

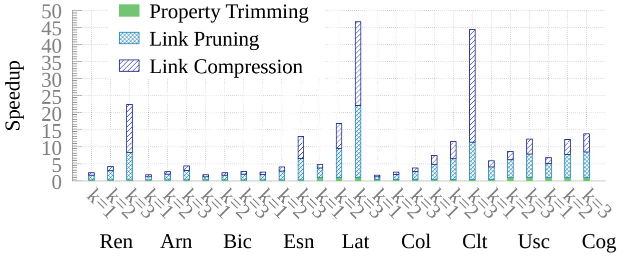


Fig. 7. Speedup in the total verification of *VeriBoost* applied to Minesweeper for BGP datasets when progressively applying *Property Trimming*, *Link Pruning*, and *Link Compression*.

Overhead. Fig. 8 displays the time for *Link Pruning*, *Link Compression* and *Property Trimming* for 260 WAN datasets. The total time of the three steps is under $510ms$, all within the millisecond range. Therefore, the time for all three steps can be considered negligible compared to the seconds-level time for verification.

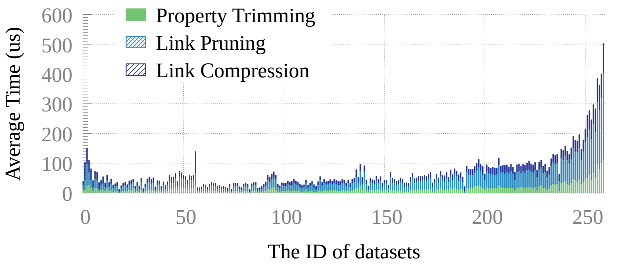


Fig. 8. The average calculation time for each step: *Link Pruning*, *Link Compression* and *Property Trimming*.

Table 2. The number of satisfied properties of Minesweeper, NetSMT, *VeriBoost*.

Network	BGP									OSPF		
	Ren	Arn	Bic	Esn	Lat	Col	Clt	Usc	Cog	Bic	Col	Usc
k=1	94	150	150	112	62	140	100	112	150	151	140	112
k=2	44	100	100	62	12	90	50	62	100	100	90	62
k=3	12	50	50	12	0	40	0	12	50	50	40	12

B Case Study

Fig. 9 shows that *VeriBoost* indeed identifies a large number of irrelevant and equivalent links for *Reachability(Greensboro, Atlanta, 200.4.219.0/24, 3)* on the real large topology of Uscarrier of 158 nodes. For this property, *VeriBoost* identifies the number of 63 irrelevant and 65 equivalent links, and the number of symbolic links is 61.

For $k = 3$, verifiers only need to check $C(61, 3) = O(10^4)$ *failure scenarios*, instead of $C(189, 3) = O(10^6)$. Finally, this property is checked for violation when "Salisbury−Concord" and "Burlington−Durham" fail.

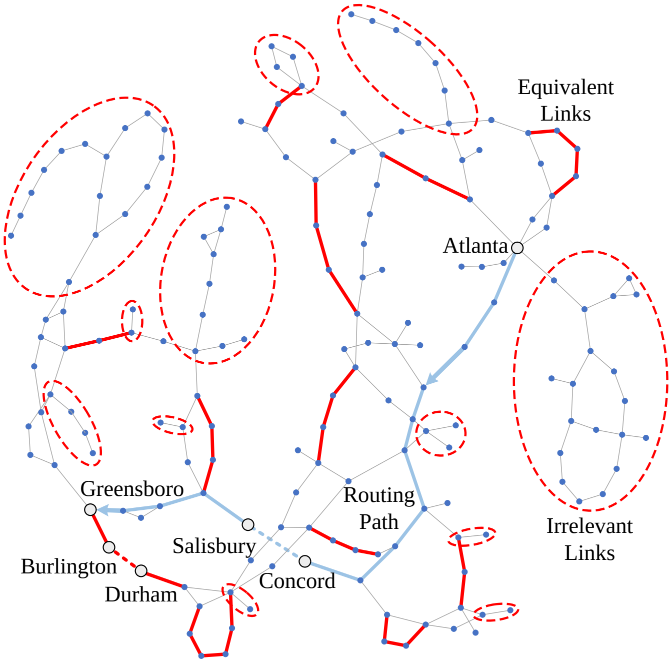


Fig. 9. An example of checking *Reachability(Greensboro, Atlanta, 200.4.219.0/24, 3)* on Uscarrier, a WAN topology from the Topology Zoo. The *irrelevant links* and *equivalent links* identified by *VeriBoost* are partially shown.

C Preliminary

Since the descriptions in Theorem 1 and 3 are not specific, such as the definition of $\mathcal{R}$ (i.e., the best routes on forwarding paths and routing paths), we define the best route, as well as forwarding paths and routing paths, in this section. The proofs of Theorem 1 and 3 are then based on these definitions.

For the property $\mathit{property}(v_1, v_2, d, k)$, since the prefix d is directly connected to v_2, the path from v_2 to v_1 is equivalent to the path from d to v_1. Therefore, in the following discussion, we omit the parameter v_2. Additionally, since our algorithm is valid for any tolerance k, we also omit the parameter k.

Definition 10. *We define several concepts related to best routes, inspired by [5, 26]. All concepts are defined under a concrete failure scenario f.*

- *$\perp$ denotes the absence of routing announcements.*
- *a_d denotes the initial routing announcement advertised by the destination d.*
- *$N(v)$ returns the set of peering neighbors of device v.*
- *$\textsc{Trs}_f(l, \mathcal{L}(u))$ returns the routing announcement after applying the export policies of the routing protocol on node u, transmitting over the link $l = (v, u)$, and applying the import policies of the routing protocol on node v. If the link l fails, the result is $\perp$.*
- *$\textsc{Sel}_f(A)$ returns the best routing announcement from a set of routing announcements according to the protocol decision, where A is a set of route announcements.*
- *$\textsc{Can}_f(v)$ returns all routes announced by neighbors, i.e., $\{\textsc{Trs}_f(l, \mathcal{L}_f(u, d)) \mid l = (v, u), u \in N(v)\}$.*
- *$\mathcal{L}_f(v, d)$ denotes the best routes to prefix d on device v.*

$$\mathcal{L}_f(v,d) = \begin{cases} a_d & v = d \\ \perp & \textsc{Can}_f(v) = \emptyset \\ \textsc{Sel}_f(\textsc{Can}_f(v)) & \textsc{Can}_f(v) \neq \emptyset \end{cases} \tag{7}$$

Definition 11. *We define some functions related to paths.*

- *$\textsc{Fwd}_f(v, d)$ returns the devices on the forwarding path from the source device v to prefix d under a concrete failure scenario f.*
- *$\textsc{Rte}_f(v, d)$ returns the devices on the routing path from the prefix d to the destination device v under a concrete failure scenario f. Note that the direction of the routing path is opposite to the direction of packet forwarding.*
- *$\textsc{Nsp}(v, d)$ returns the devices that don't appear on any simple path from v to d.*
- *$\textsc{Atp}(v, d)$ returns the set of all trivial paths from v to d.*

With Definition 10 and Definition 11, for the property $\mathit{property}(v_1, v_2, d, k)$, we have:

$$\mathcal{R}_f(v_1, v_2, d) = \{\mathcal{L}_f(v, d) \mid v \in \textsc{Fwd}_f(v_1, d) \cup \textsc{Rte}_f(v_1, d)\} \tag{8}$$

With the above Definition 10 and 11, we also restate our two assumptions in §4.2 as:

Assumption 1. *The BGP protocol converges to a unique state, i.e., for* $\forall d$*, if* $v = u$*, then* $\mathcal{L}(v, d) = \mathcal{L}(u, d)$*.*

Assumption 1 holds true in most practical scenarios as demonstrated in [14,29]. If this assumption does not hold, the best routes of BGP may exhibit randomness, meaning that the value of $\mathcal{L}(v, d)$ will be random.

Assumption 2. *The network exhibits monotonicity, meaning that the weight of a path does not decrease when prefixed by a link. Specifically, suppose* a *is a routing announcement. For* $\forall l$*, it holds that* $\textsc{Trs}_f(l, a) \neq \textsc{Sel}_f(\{a, \textsc{Trs}_f(l, a)\})$*.*

Assumption 2 generally holds in BGP and OSPF [22,28].

D The Proof of Theorem 1

We prove Theorem 1 with following Lemmas.

Lemma 1. *For* $T = (V, L)$ *and property*(v_1, v_2, d, k)*, if a link* $l \in \textsc{Nsp}(v_1, d)$*, then* $\forall f = (L_{\mathcal{U}}, L_{\mathcal{D}})$*, where* $l \in L_{\mathcal{U}}$*, and* $\forall v \in \textsc{Rte}_f(v_1, d)$*, we have:*

$$\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$$

, where $f' = (L_{\mathcal{U}} \setminus \{l\}, L_{\mathcal{D}} \cup \{l\})$*.*

Proof. We prove it by mathematical induction, which works by the number of routing spreading hops.

Base case. d is connected routes on v. In this case, whether the link fails or not, v still receives the routes for d through the link (v, d). According to Equation 7, we have $\mathcal{L}_f(v, d) = a_d$ and $\mathcal{L}_{f'}(v, d) = a_d$, then we have $\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$.

Step case. Suppose u is the routing last hop of v under f, which send the best route announcement $\mathcal{L}_f(u, d)$ to v through link $l' = (u, v)$, and we denote the best route announcement $\textsc{Trs}_f(l', \mathcal{L}_f(u, d))$ on v as a. There are two cases:

Case (i) $l' \notin \textsc{Nsp}(v_1, d)$ (①); Because of the definition of f', then f' differs from f by only the links in $\textsc{Nsp}_f(v_1, d)$, and because of ①, then $l' \in f'.L_{\mathcal{U}}$ (②); Because $\textsc{Sel}_f$ and $\textsc{Sel}_{f'}$ is based on the same best routing decision, then $\textsc{Sel}_f = \textsc{Sel}_{f'}$ (③); Because of $l' \in f.L_{\mathcal{U}}$ and ②, and the same import and export policies under f and f', then $\textsc{Trs}_f(l', \mathcal{L}_f(u, d)) = \textsc{Trs}_{f'}(l', \mathcal{L}_f(u, d))$ (④); By the induction hypothesis that $i-1$ holds, then $\mathcal{L}_f(u, d) = \mathcal{L}_{f'}(u, d)$ (⑤); Because of ④ and ⑤, then $\textsc{Trs}_f(l', \mathcal{L}_f(u, d)) = \textsc{Trs}_{f'}(l', \mathcal{L}_{f'}(u, d))$ (⑥); Because $a = \textsc{Trs}_f(l', \mathcal{L}_f(u, d))$ and ⑥, then $a \in \textsc{Can}_{f'}(v)$ (⑦); Because $f'.L_{\mathcal{U}}$ only has fewer links in $\textsc{Nsp}(v_1, d)$ compared to $f.L_{\mathcal{U}}$, and ①, then $\textsc{Can}_{f'}(v) \subset \textsc{Can}_f(v)$ (⑧); Because of ⑦⑧, we know that $a \notin \textsc{Can}_f(v) \setminus \textsc{Can}_{f'}(v)$ (⑨); Because of ③⑨, and Assumption 1, we know that $\textsc{Sel}_f(\textsc{Can}_f(v)) = \textsc{Sel}_f(\textsc{Can}_{f'}(v)) = \textsc{Sel}_{f'}(\textsc{Can}_{f'}(v))$, i.e., $\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$.

Case (ii) $l' \in \textsc{Nsp}(v_1, d)$ (①). We prove by contradiction that this case does not occur.

Because of ①, $\mathcal{L}_f(v, d)$ is spread by a simple path $path = \{v\text{-}(v, u)\text{-}u\text{-}\ldots\text{-}v\text{-}v'\text{-}\ldots\text{-}d\}$ (②), i.e., the routing announcement passes through v twice; Because of ②, a is the routing announcement that arrives at v for the second time, with *path* (③); Then, we let a' be the first time the routing announcement arrives at v, with $path' = \{v\text{-}v'\text{-}\ldots\text{-}d\}$ (④); By comparing the two paths from ③ and ④, we have $a = \textsc{Trs}_f((v, u), \textsc{Trs}_f(..., \textsc{Trs}_f(..., a')))$ (⑤); Because a and a' are the routing announcements that arrive at v, then $\{a, a'\} \in \textsc{Can}_f(v)$ (⑥); Because of Assumption 2, and ⑤⑥, we have $a \neq \textsc{Sel}_f(\textsc{Can}_f(v))$, i.e., a isn't the best route on f, which contradicts the fact that a is the best route.

Lemma 2. *For $T = (V, L)$ and property(v_1, v_2, d, k), if a link $l \in \textsc{Nsp}(v_1, d)$, then $\forall f = (L_{\mathcal{U}}, L_{\mathcal{D}})$, where $l \in L_{\mathcal{U}}$, and $\forall v \in \textsc{Fwd}_f(v_1, d)$, we have:*

$$\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$$

, where $f' = (L_{\mathcal{U}} \setminus \{l\}, L_{\mathcal{D}} \cup \{l\})$.

Since the proof of Lemma 2 is similar to the proof of Lemma 2, we omit it.
With Lemma 1, Lemma 2 and Equation 8, we know Theorem 1 holds.

E The Proof of Theorem 2

Proof. Because Line 10 of Alg. 1 returns the complement set of the links in Lines 2-9, we only need to prove that Lines 2-9 include the links of all *simple paths* on T. Suppose a link l appeared on a *simple path* $path_T$, but Lines 2-9 exclude it. Let n' be the node of v_1 on $\mathcal{T}$, n'' be the node of v_2 on $\mathcal{T}$, and $path_{\mathcal{T}}$ be the corresponding path of $path_T$ on $\mathcal{T}$. Because (1) $\mathcal{T}$ is a tree and (2) Lines 6-9 include all links within the PBCs from n' to n'', Line 6-9 excludes link l only under one condition: Line 6 excludes $path_{\mathcal{T}}$ with the format $\left\{n'\text{-}...\text{-}n^{\theta}\text{-}n^{v}\text{-}...\text{-}n^{v}\text{-}n^{\theta}\text{-}...\text{-}n''\right\}$ or $\left\{n'\text{-}...\text{-}n^{v}\text{-}n^{\theta}\text{-}...\text{-}n^{\theta}\text{-}n^{v}\text{-}...\text{-}n''\right\}$. In either case, traversal from n' to n'' must pass through n^v on $\mathcal{T}$ twice, so $path_T$ starting from v_1 to v_2 must also pass through v on T twice. This contradicts the fact that $path_T$ is a *simple path.*

F The Proof of Theorem 3

We prove Theorem 3 with following Lemmas.

Lemma 3. *For $T = (V, L)$ and property(v_1, v_2, d, k), if a link $l \in tp$, $tp \in \textsc{Atp}(v_1, d)$, then $\forall f = (L_{\mathcal{U}}, L_{\mathcal{D}})$, where $l \notin L_{\mathcal{U}}$, and $\forall v$, where v is the endpoint of any trivial path, we have:*

$$\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$$

, where $f' = (L_{\mathcal{U}} \setminus \{l'\} \cup \{l\}, L_{\mathcal{D}} \cup \{l'\} \setminus \{l\}), l' \in tp, l' \neq l$.

Proof. We prove it by mathematical induction. The route announcement of d propagates through a sequence of trivial paths. Our induction works by the number of trivial paths from d to the endpoint v.

Base case. d is connected routes on v. In this case, whether the link fails or not, v still receive the routes for d through the link (v, d). According to Equation 7, we have $\mathcal{L}_f(v, d) = a_d$ and $\mathcal{L}_{f'}(v, d) = a_d$, then we have $\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$.

Step case. Assume that the path from d to v passes through i trivial paths, where $i \neq 0$. We denote the set of endpoints whose path from d to v contains exactly $i-1$ trivial paths as $E(v)$. We denote the set of all trivial paths between v and the endpoints in $E(v)$ as TP. There are two cases:

Case (i) $tp \notin TP$ (①); Because of ①, then $tp \in \textsc{Atp}(u, d)$, where $u \in E(v)$ (②); Because the inductive hypothesis holds for the $i-1$ case and ②, then $\forall u \in E(v)$, $\mathcal{L}_f(u, d) = \mathcal{L}_{f'}(u, d)$ (③); Because $\textsc{Sel}_f$ and $\textsc{Sel}_{f'}$ are based on the same best routing decision, then $\textsc{Sel}_f = \textsc{Sel}_{f'}$ (④); Because all import and export policies are the same under f and f', and the states of all links from $E(v)$ to v are the same, e.g., failed or not, then all $\textsc{Trs}_f$ and $\textsc{Trs}_{f'}$ between u and $E(v)$ are the same (⑤). Because of ③④⑤, then $\forall u \in E(v)$, $\textsc{Trs}_f((v, ...), \textsc{Trs}_f(..., \textsc{Trs}_f(\mathcal{L}_f(u, d))))$ is equal to $\textsc{Trs}_{f'}((v, ...), \textsc{Trs}_{f'}(..., \textsc{Trs}_{f'}(\mathcal{L}_{f'}(u, d))))$, i.e., $\textsc{Can}_f(v, d) = \textsc{Can}_{f'}(v, d)$ (⑥). Because of ④⑥, then $\textsc{Sel}_f(\textsc{Can}_f(v)) = \textsc{Sel}_f(\textsc{Can}_{f'}(v)) = \textsc{Sel}_{f'}(\textsc{Can}_{f'}(v))$, i.e., $\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$.

Case (ii) $tp \in TP$ (①). Since the states of all links from d to $E(v)$ are identical, and the import and export policies as well as the initial routing are the same under both f and f', we have $\forall u \in E(v)$, $\mathcal{L}_f(u, d) = \mathcal{L}_{f'}(u, d)$ (②), and this conclusion is the same as in *Case (i)*, but it relies on Assumption 1 rather than the inductive hypothesis; Because v is one end points of *tp*, we then let u be the another end points of *tp*, and $a^l = \textsc{Trs}_f(..., \textsc{Trs}_f(l, ..., \textsc{Trs}_f(\mathcal{L}_f(u, d)))$ (③), and $a^{l'} = \textsc{Trs}_{f'}(..., \textsc{Trs}_{f'}(l', ..., \textsc{Trs}_{f'}(\mathcal{L}_f(u, d)))$ (④); With the same method in *Case (i)*, we could prove that $\textsc{Can}_f(v, d) \backslash a^l = \textsc{Can}_{f'}(v, d) \backslash a^{l'}$ (⑤); Because $l \in f.\mathcal{L}_\mathfrak{D}$ and ③, then $a^l = \perp$ (⑥); Because $l' \in f'.\mathcal{L}_\mathfrak{D}$ and ④, then $a^{l'} = \perp$ (⑦); Because of ⑤⑥⑦, $\textsc{Sel}_f(\textsc{Can}_f(v, d)) = \textsc{Sel}_f(\textsc{Can}_f(v, d) \backslash a^l) = \textsc{Sel}_f(\textsc{Can}_{f'}(v, d) \backslash a^{l'}) = \textsc{Sel}_{f'}(\textsc{Can}_{f'}(v, d) \backslash a^{l'}) = \textsc{Sel}_{f'}(\textsc{Can}_{f'}(v, d))$, i.e., $\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$.

Lemma 4. *For $T = (V, L)$ and property(v_1, v_2, d, k), if a link $l \in tp$ and $tp \in$ $\textsc{Atp}(v_1, d)$, then $\forall f = (L_\mathfrak{U}, L_\mathfrak{D})$, where $l \notin L_\mathfrak{U}$, and $\forall v \in \textsc{Rte}_f(v_1, d)$, we have:*

$$\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$$

, where $f' = (L_\mathfrak{U} \setminus \{l'\} \cup \{l\}, L_\mathfrak{D} \cup \{l'\} \setminus \{l\}), l' \in tp, l' \neq l$.

Proof. We prove this by leveraging Lemma 3. Suppose u is the endpoint of the last trivial path tp' of v under f, which sends the best route announcement $\mathcal{L}_f(u)$ to v through tp'. We now prove that the node u' belongs to the part of $\textsc{Rte}_f(v_1, d)$ from u to v that satisfies $\mathcal{L}_f(u', d) = \mathcal{L}_{f'}(u', d)$. Similarly, the node u' belongs to the part of $\textsc{Rte}_f(v_1, d)$ from d to u can be proven to satisfy this in the same manner.

Case (i) $tp = tp'$. We prove that this situation does not occur. Since $l \in tp$, it follows that $l \in tp'$. Given that $l \in f.\mathcal{L}_\mathfrak{D}$, tp' is blocked. This contradicts the assumption that tp' sends the best route announcement.

Case (ii) $tp \neq tp'$. According to Lemma 3, we know that $\mathcal{L}_f(u, d) = \mathcal{L}_{f'}(u, d)$ and $\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$, i.e., the best routes for d at both ends of a trivial path are the same. Moreover, for $u' \in tp'$, because u' can only receive the route from either u or v, we simply prove that $\mathcal{L}_f(u', d) = \mathcal{L}_{f'}(u', d)$ with the method as in Lemma 3 *Case (i)*.

Lemma 5. *For $T = (V, L)$ and property(v_1, v_2, d, k), if a link $l \in tp$ and $tp \in$ $\textsc{Atp}(v_1, d)$, then $\forall f = (L_{\mathcal{U}}, L_{\mathcal{D}})$, where $l \notin L_{\mathcal{U}}$, and $\forall v \in \textsc{Fwd}_f(v_1, d)$, we have:*

$$\mathcal{L}_f(v, d) = \mathcal{L}_{f'}(v, d)$$

, where $f' = (L_{\mathcal{U}} \setminus \{l'\} \cup \{l\}, L_{\mathcal{D}} \cup \{l'\} \setminus \{l\}), l' \in tp, l' \neq l$.

Since the proof of Lemma 5 is similar to the proof of Lemma 2, we omit it. With Lemma 4, 5 and Equation 8, Theorem 1 holds.