Comprehensive Network Configuration Verification via Effective Environment Reduction

Xinzhe Liu*‡, Yahui Li*†, Han Zhang*‡, Renrui Tian*‡, Xia Yin*‡, Xingang Shi*‡,
Gang Ren*‡, Jilong Wang*‡ and Jiangyuan Yao§†
*Tsinghua University, †State Key Laboratory of Internet Architecture, ‡Zhongguancun Laboratory, §Hainan University

IEEE INFOCOM 2026 - IEEE Conference on Computer Communications | 979-8-3315-4961-9/26/$31.00 ©2026 IEEE | DOI: 10.1109/INFOCOM59046.2026.11571533

***Abstract*—Verifying network configurations under all possible environments is critical for uncovering latent configuration bugs. However, due to the vastness of the environment space and the limitations of existing approaches, current tools either fail to scale or explore only a tiny fraction of the full environment space, missing many potential issues. In this paper, we show that it is unnecessary to explore the full environment space. Instead, we introduce an effective reduction of the environment space, under which the network preserves all routing and forwarding behaviors exhibited under the full environment space. We develop a theory of network behavior equivalence between the full and effectively reduced environment spaces, and propose an efficient algorithm to compute such an effective reduction. We implement our approach in a tool called Bulwark. Experiments on both real-world and synthetic networks show that Bulwark uncovers numerous new misconfigurations and delivers orders-of-magnitude speedups over state-of-the-art verifiers.**

***Index Terms*—network reliability, network environments**

I. Introduction

The distributed network is a complex system whose behavior (i.e., how routing messages are processed and how paths for traffic are selected) depends critically on the configuration of every router. Any misconfigurations can lead to large-scale network disruptions and severe financial losses, often making headlines [1]–[5].

Many configuration bugs only manifest under specific environments–i.e., when the network experiences specific link states and receives particular sets of routing messages from external neighbors. Therefore, it is crucial to verify the configurations across all possible environments to ensure the network's correctness.

However, verification under all such environments is particularly challenging due to the vastness of environment space:

- **Exponential link states:** Each link in the network can be either up or down. For a network with K links, this results in 2^K possible link states. Each of these failure scenarios can alter the network's behavior and potentially expose latent configuration bugs.
- **Unbounded external advertisement states:** The network may receive different sets of routing messages from each external neighbor. Let T_r denote the total number of possible routing messages; then each neighbor can advertise up to 2^{T_r} different subsets. With T_e external neighbors, the total number of external advertisement states grows exponentially to $2^{T_r \cdot T_e}$. In practice, T_r is often unbounded. For example, suppose each BGP routing message includes an IPv4 prefix, a standard community list, and an AS path. There are up to $2^{33}-1$ possible IPv4 prefixes, $2^{2^{32}}$ possible community lists, and a nearly unbounded number of AS paths due to the vast AS number space and their arbitrary concatenations. As a result, the routing message space is essentially infinite.

The sheer size of environment space, driven primarily by unbounded external advertisement states, introduces a fundamental trade-off between the number of environments explored and the overall scalability of the verification. It is hard to explore the majority of the full environment space with high scalability–after all, any non-zero fraction of an infinite set remains infinite. However, existing control plane verifiers [6]–[23] commonly aim to balance both goals by employing increasingly sophisticated techniques to explore as much of the space as possible. Consequently, they typically fall into two categories: (i) Some of them [13]–[18] prioritize scalability by analyzing only a tiny subset of the full environment space, which inevitably risks missing critical configuration bugs. (ii) Others [7] aim for exhaustive coverage of all possible environments but often suffer from severe scalability issues, making them impractical for real-world networks.

Unlike the principle employed by prior works, this paper introduces a fundamentally different approach. We prove that it is unnecessary to verify the network under the full environment space. Instead, there exists a small subset of the full environment space–termed the *effective reduction*–under which the network's routing and forwarding behaviors are equivalent to those under the full environment space. This enables us to reduce the infinite environment space into a small and representative subset by specifically targeting the unbounded external advertisement states, making scalable and sound verification under all possible environments feasible.

More specifically, we begin by defining a formal network model that captures the core operations of routers in processing routing messages and forwarding packets. This model also incorporates the full environment space, which encodes all possible combinations of link states and external advertisement states. We further define the formal semantics of the model, which precisely describe how routers interact and how network behavior evolves under different environments.

Given this model, we turn to the process of characterizing the environment reductions. Intuitively, we would like to define

Yahui Li is the corresponding author. This work was supported by the National Key R&D Program of China 2024YFB2906800 and the National Natural Science Foundation of China 62394322.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

reductions that preserve the equivalent routing and forwarding behaviors of the network under the full environment space. However, doing so directly is challenging because validating that network behaviors under two environment spaces are equivalent is as hard as the verification under the full environment space! We address this challenge by defining a class of environment reductions, termed *effective environment reductions*. These reductions are characterized by several conditions that can be checked efficiently without a global simulation under every environment. Our core theoretical result shows that these effective conditions imply the network behavior equivalence (i.e., routing and forwarding behavior equivalence) between the full and effectively reduced spaces. Furthermore, we present a practical algorithm for computing such an effective reduction from network configurations.

Although the full environment space is reduced to a small subset, computing the network behaviors under the reduced space can still be computationally expensive. To address this, we adopt the idea of symbolic execution and extend the formal network model into a symbolic form, which enables simulating all possible network behaviors across the reduced space through a single symbolic execution.

We implement our ideas in a tool called Bulwark and evaluate its effectiveness and performance on a diverse set of real-world and synthetic networks. In our experiments, Bulwark successfully identifies 119 misconfigurations in real network scenarios (e.g., Internet2 [24]) that could lead to severe issues such as prefix hijacks and route leaks. Notably, 110 of them are previously unknown. Moreover, Bulwark consistently outperforms state-of-the-art verifiers across networks of varying sizes, achieving an approximate $100\times$ speedup.

Contribution: Our contributions are summarized as follows:

- A novel principle for scaling verification under all possible environments: instead of exhaustively exploring them, it suffices to verify under a small and sound subset, termed *effective reduction*.
- A theory of network behavior equivalence between the full environment space and its effective reduction, and an efficient algorithm for computing such a reduction.
- A tool Bulwark built on our ideas and a comprehensive evaluation of Bulwark on various synthetic and real-world networks, demonstrating its effectiveness and efficiency.

II. Related work

Based on their coverage of the environment space, existing control plane verifiers can be classified into three categories:

Verification under a single environment. Tools such as Batfish [6] simulate the network behavior under a specific environment. While these tools offer high scalability, they fall short in handling multiple environments, as each additional environment requires a separate simulation. This limitation results in many configuration bugs remaining undetected.

Verification under multiple environments: Several tools [13]–[16] leverage techniques such as symbolic simulation and graph abstraction to exhaustively explore environments with all possible link states. However, they typically assume a fixed external advertisement state. Due to the unbounded nature of the external advertisement state space, they do not scale to verify the network behavior across all possible external advertisement states. Following the same verification principle (i.e., exhaustive exploration), tools such as [17], [18] attempt to extend symbolic simulation to explore environments with diverse external advertisement states (under a fixed link state). However, to remain scalable, they are forced to restrict exploration to only a tiny fraction of the external advertisement state space. For instance, Expresso [17] only explores environments where external neighbors either send all possible messages with a given prefix or send none of them, thereby covering fewer than $2^{(1-2^{2^{32}})(2^{33}-1)T_e}$ of all possible external advertisement states.

Verification under all possible environments. To the best of our knowledge, Minesweeper [7] is the only verifier that attempts to verify the network across all possible environments. It encodes the network configuration and properties to be verified as SMT constraints and employs an SMT solver to check their satisfiability under all possible environments. However, even state-of-the-art SMT solvers like Z3 [25] struggle to solve such complex expressions, severely limiting the practical scalability of Minesweeper.

Data plane verifiers [26]–[32], outlier detection tools [33]–[35], and LLM-based approaches [36], [37] are also limited to verifying the network under a single, fixed environment.

III. Network Model

As shown in Fig. 1, the network instance is defined as a tuple $N = (G, M, ENV, \mathsf{originate}, \mathsf{trans}, \oplus, \mathsf{nexthop})$.

- $G = (V, E)$ is a graph where V is the set of router vertices, and E is the set of directed links between routers. The vertex set V consists of V_I and V_E, representing internal routers and external neighbors, respectively.
- $M = Prefix \times Attr_1 \times ... \times Attr_n$ represents the set of routing messages across different routing protocols (e.g., ISIS, BGP). Here, $Prefix$ denotes the set of all prefixes. Each $Attr_i$ represents the set of values for a specific attribute, which varies across different protocols.
- $ENV = \mathcal{L} \times \mathcal{N}$ denotes the set of all possible environments the network may experience. Here, $\mathcal{L}$ contains a set of link-state functions l, capturing all possible link states of the network. Each l maps every link $e \in E$ to a boolean value indicating whether the link is up (True) or down (False). $\mathcal{N}$ captures all possible external advertisement states, in which different external neighbors may advertise different sets of routing messages to the network. It consists of a set of advertisement-state functions n, where each function n maps each external neighbor $v_e \in V_E$ to a set of routing messages that v_e advertises under a particular state. In the full environment space, the value of $n(v_e)$ is drawn from the power set of all possible routing messages, which is defined as the Cartesian product of the complete domains of prefixes and routing attributes.
- originate is the function that specifies the set of routing messages initially generated by each internal router

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

Network instance: $\boxed{N = (G, M, ENV, \mathsf{originate}, \mathsf{trans}, \oplus, \mathsf{nexthop})}$

G	$=$	(V, E)	*network topology*
V_I			*internal router vertices*
V_E			*external neighbor vertices*
V	$=$	$V_I \cup V_E$	*router vertices*
E	$=$	$V \times V$	*links*
$Prefix$	$=$	$IP \times MaskLength$	*prefixes*
M	$=$	$Prefix \times Attr_1 \times ... \times Attr_n$	*routing messages*
$\mathcal{L}$	$=$	$\{l \mid l : E \to \mathsf{True/False}\}$	*link states*
$\mathcal{N}$	$=$	$\{n \mid n : V_E \to 2^{Prefix \times Attr_1 \times ... \times Attr_n}\}$	*external advertisement states*
ENV	$=$	$\mathcal{L} \times \mathcal{N}$	*environments*
$\mathsf{originate}$	$:$	$V_I \to 2^M$	*message originate function*
trans	$:$	$E \times M \to M$	*message transfer function*
$\oplus$	$:$	$M \times M \to M$	*message merge function*
$\mathsf{nexthop}$	$:$	$IP \times 2^M \to V \cup \{\perp\}$	*next-hop decision function*

Network solution: $\boxed{\mathcal{S} : ENV \to (V \to 2^M)}$

$$\tau : \mathcal{T} \to 2^V$$

$$\mathcal{S}^t(env)(v) = \begin{cases} \mathsf{originate}(v) & t = 0, v \in V_I \\ n(v) & t = 0, v \in V_E \\ \mathcal{S}^{t-1}(env)(v) & t > 0, v \notin \tau(t) \\ \mathcal{S}^0(env)(v) \oplus (\bigoplus_{e=(u,v) \in E} \mathsf{choices}^t(env, e)) & t > 0, v \in \tau(t) \end{cases}$$

$$\mathsf{choices}^t(env, e) = \begin{cases} \mathsf{trans}_e(\mathcal{S}^{t-1}(env)(u)) & l(e = (u, v)) = \mathsf{True} \\ \emptyset & l(e = (u, v)) = \mathsf{False} \end{cases}$$

$$\mathsf{fwd}(env, v, ip) = \mathsf{nexthop}(ip, \mathcal{S}(env)(v))$$

Fig. 1: Formal network model and its semantics.

$v_i \in V_I$. Specifically, $\mathsf{originate}(v_i)$ contains the routing messages whose prefixes are directly connected to v_i.

- trans is the transfer function that describes how routing messages are modified (or filtered) as they propagate between router vertices. For link $e = (u, v)$ and routing message $m \in M$, trans_e may modify the attributes carried by m or filter the message.
- $\oplus$ is a merge function that models the selection process for routing messages with the same destination prefix by comparing their attributes.
- $\mathsf{nexthop}$ is a function that selects a router $v \in V$ as the next hop for a given IP address $ip \in IP$, based on the longest-prefix match over a given set of routing messages. If no matching message is found, the function returns $\perp$, indicating that the destination is unreachable.

Given a formal network N, we define its stable solution as a function $\mathcal{S} : ENV \to (V \to 2^M)$, which maps each environment $env \in ENV$ to a function that assigns each router $v \in V$ the set of routing messages it selects under env. The computation of $\mathcal{S}$ follows the formal semantics of the model, as illustrated at the bottom of Fig. 1. Here, $\tau : \mathcal{T} \to 2^V$ is a time schedule, which maps each time step to the set of routers that are scheduled to process routing messages at that step. And $\mathcal{S}^t$ denotes the intermediate solution at time step t. At the initial step $t = 0$ under environment $env = (l, n)$, each internal router $v_i \in V_I$ initializes its solution with the set of messages it originates, i.e., $\mathsf{originate}(v_i)$. Each external neighbor $v_e \in V_E$ initializes its solution with the messages it advertises under env, i.e., $n(v_e)$. At each subsequent time step $t > 0$, if a vertex v is not scheduled at time t (i.e., $v \notin \tau(t)$), its solution remains unchanged. Otherwise, v merges the set of messages received from each neighbor u connected via link $e = (u, v)$. The set of received messages, denoted $\mathsf{choices}^t(env, e)$, depends on the state of e under env. If the link is up (i.e., $l(e) = \mathsf{True}$), the messages are obtained by applying the transfer function trans_e to the messages in $\mathcal{S}^{t-1}(env)(u)$. If the link is down (i.e., $l(e) = \mathsf{False}$), then the message set is empty. This process repeats until a fixed point is reached for all environments, i.e., for all $env \in ENV$ and $v \in V$, $\mathcal{S}^t(env)(v) = \mathcal{S}^{t-1}(env)(v)$. Alternatively, based on the prefixes carried in routing messages, the stable solution can also be represented as $\mathcal{S}' : ENV \to (V \to (Prefix \to M))$.

Given the stable solution $\mathcal{S}$, the network's forwarding behavior under every environment can be directly determined. We define $\mathsf{fwd}(env, v, ip)$ as the next hop router selected by router v for forwarding traffic destined to IP address $ip \in IP$, based on its selected routing messages under environment env.

IV. Effective Environment Reduction

We now build on the formal network model to describe the theory and implications of effective environment reductions.

Environment reductions. We start by formalizing the notion of environment reductions. An environment reduction relates two environment spaces: the full environment space $ENV =$

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

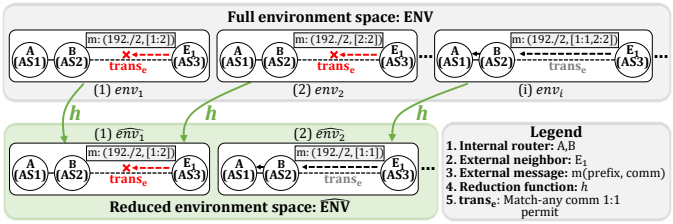


Fig. 2: An illustrative example of environment reduction.

$\mathcal{L} \times \mathcal{N}$, and the reduced environment space $\widehat{ENV} = \mathcal{L} \times \widehat{\mathcal{N}}$, via a reduction function h for external advertisement states. The reduction function h is defined as a tuple of functions $h = (h_p, h_{a_1}, \ldots, h_{a_n})$, where $h_p : Prefix \to \widehat{Prefix}$ maps each prefix to its reduced representative, and $h_{a_i} : Attr_i \to \widehat{Attr_i}$ maps each attribute value to a reduced one. Here, $\widehat{Prefix} \subseteq Prefix$ and $\widehat{Attr_i} \subseteq Attr_i$. The function h operates on each advertisement-state function $n \in \mathcal{N}$, yielding the reduced external advertisement state space: $h(\mathcal{N}) = \widehat{\mathcal{N}} = \{\hat{n} \mid \hat{n} : V_E \to 2^{\widehat{Prefix} \times \widehat{Attr_1} \times \cdots \times \widehat{Attr_n}}\}$. Fig. 2 illustrates an example of an environment reduction.

A. Effective reduction conditions

In an environment reduction, h can be arbitrary reduction functions, but we are only interested in reductions that preserve both routing and forwarding behaviors of the network under the full environment space. Therefore, we formulate a set of easy-to-check conditions on environment reductions that imply network behavior equivalence. These conditions are restrictions on the function h (i.e., the prefix reduction function h_p and attribute reduction functions h_{a_i}). *Effective environment reductions* are those that satisfy these conditions.

Definition 1 (Originating Effectiveness)**.** *Given a network N, a prefix reduction function h_p is originating effective if: For every prefix $prefix \in Prefix$, if $prefix$ originates from an internal router $v_i \in V_I$, then (1) $h_p(prefix) = prefix$; and (2) there does not exist any other prefix $prefix_i \in Prefix$ such that $prefix_i \neq prefix$ and $h_p(prefix_i) = h_p(prefix)$.*

Definition 2 (Aggregating Effectiveness)**.** *Given a network N, a prefix reduction function h_p is aggregating effective if: For every prefix $prefix \in Prefix$, if $prefix$ involves route aggregation, then (1) $h_p(prefix) = prefix$; and (2) there does not exist any other prefix $prefix_i \in Prefix$ such that $prefix_i \neq prefix$ and $h_p(prefix_i) = h_p(prefix)$.*

Definition 3 (Sub-prefix Effectiveness)**.** *Given a network N, a prefix reduction function h_p is sub-prefix effective if: For every two prefixes $prefix_1, prefix_2 \in Prefix$, if $prefix_2$ is the sub-prefix of $prefix_1$, then $h_p(prefix_2)$ is the sub-prefix of $h_p(prefix_1)$.*

Definition 4 (Policy Effectiveness)**.** *Given a network N, a prefix reduction function h_p or attribute reduction function h_{a_i} is policy effective if: For every two routing messages m_1 and m_2 that differ only in the prefix or attribute $Attr_i$–m_1 carries $prefix \in Prefix$ and m_2 carries $h_p(prefix)$, or m_1 carries $attr \in Attr_i$ and m_2 carries $h_{a_i}(attr)$–and for every path p starting from each external neighbor $v_e \in V_E$, the transfer functions* trans *on p always apply identical operations to m_1 and m_2 (i.e., the same permit/deny decisions and attribute modification) when m_1 and m_2 propagate along p.*

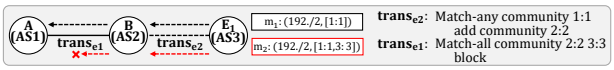


Fig. 3: An illustrative example of policy effectiveness.

As shown in Fig. 3, messages from E_1 carrying communities [1:1, 3:3] will be blocked by $trans_{e1}$, unlike those carrying only [1:1]. To ensure policy effectiveness, the community reduction function $h_{a_{com}}$ must map them to different values.

Definition 5 (Priority Effectiveness)**.** *Given a network N, an attribute reduction function h_{a_i} is priority effective if: For every two routing messages m_1 and m_2 that differ only in the attribute $Attr_i$ (i.e., m_1 carries $attr \in Attr_i$ and m_2 carries $h_{a_i}(attr)$), m_1 and m_2 have the same priority during the message selection process.*

B. Effectiveness implies network behavior equivalence

Definition 6 (Network Behavior Equivalence)**.** *The network behaviors under environment spaces ENV_1 and ENV_2 are equivalent iff:*

- **Routing behavior equivalence***: For every prefix $prefix \in Prefix$ and every environment $env \in ENV_1$, there exists an environment $env' \in ENV_2$ and a prefix $prefix' \in Prefix$ such that, for every router $v \in V$, the propagation path of the $\mathcal{S}'(env)(v)(prefix)$ is identical to that of $\mathcal{S}'(env')(v)(prefix')$. And the same condition holds in the reverse direction.*
- **Forwarding behavior equivalence***: For every IP address ip covered by $prefix \in Prefix$ and every environment $env \in ENV_1$, there exists an IP address ip' covered by $prefix'$ and an environment $env' \in ENV_2$ such that, for every router $v \in V$, $\mathsf{fwd}(env, v, ip) = \mathsf{fwd}(env', v, ip')$. And the same condition holds in the reverse direction.*

Lemma 1. *Let h_{a_i} be an effective attribute reduction function over $Attr_i$. Network behaviors under the full space ENV and the space $\widehat{ENV}_{Attr_i}$ reduced by h_{a_i} are equivalent.*

Proof. Define APC_j^i as the set of values $attr_{(i,y)} \in Attr_i$ that are mapped to the same reduced value $\widehat{attr}_{(i,j)} \in \widehat{Attr_i}$ by h_{a_i} (i.e., $h_{a_i}(attr_{(i,y)}) = \widehat{attr}_{(i,j)}$). Then, $Attr_i$ equals $APC_1^i \cup \cdots \cup APC_z^i$. We define the reduced environment space $\widehat{ENV}_{APC_j^i} = \mathcal{L} \times \widehat{\mathcal{N}}_{APC_j^i}$, where $\widehat{\mathcal{N}}_{APC_j^i} = \{\hat{n} \mid \hat{n} : V_E \to 2^{Prefix \times Attr_1 \times \cdots \times (APC_1^i \cup \cdots \cup \{\widehat{attr}_{(i,j)}\} \cup \cdots \cup APC_z^i) \times \cdots \times Attr_n}\}$.

We begin by proving that for every $prefix \in Prefix$, every IP ip covered by $prefix$, and every $env \in ENV$, there exists an environment $\widehat{env} \in \widehat{ENV}_{APC_j^i}$ such that, for every $v \in V$, the propagation path of $\mathcal{S}'(env)(v)(prefix)$ equals that of $\mathcal{S}'(\widehat{env})(v)(prefix)$ and $\mathsf{fwd}(env, v, ip) = \mathsf{fwd}(\widehat{env}, v, ip)$:

- **Case 1:** No external neighbor $v_e \in V_E$ sends messages with attributes in APC_j^i under env. In this case, env trivially belongs to $\widehat{ENV}_{APC_j^i}$, and setting $\widehat{env} = env$ immediately satisfies the conditions.
- **Case 2:** Some external neighbors send messages carrying attributes from APC_j^i. Let $SM_{APC_j^i}$ denote all such

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

messages sent by an external neighbor $v_e \in V_E$. For a given prefix $prefix_i$, define $SM_{APC_j^i}(prefix_i)$ as the subset of $SM_{APC_j^i}$ with prefix $prefix_i$. Since each external neighbor typically advertises at most one route per prefix, we have $SM_{APC_j^i}(prefix_i) = \{m_i = (prefix_i, attr_{(1,x)}, ..., attr_{(i,y)}, ..., attr_{(n,z)})\}$, where $attr_{(i,y)} \in APC_j^i$. We now construct a new message $\widehat{m_i} = (prefix_i, attr_{(1,x)}, \ldots, \widehat{attr}_{(i,j)}, \ldots, attr_{(n,z)})$, where $h_{a_i}(attr_{(i,y)}) = \widehat{attr}_{(i,j)}$. Let env_i be identical to env, except that v_e sends $\{\widehat{m_i}\}$ instead of $\{m_i\}$. Since h_{a_i} is *policy effective* and *priority effective*, we can conclude that for every router v and every message $m \in \mathcal{S}(env)(v)$: (i) If m is derived from m_i, there exists $m' \in \mathcal{S}(env_i)(v)$ derived from $\widehat{m_i}$ such that m' and m differ only in the attribute $Attr_i$; (ii) Otherwise, m also appears in $\mathcal{S}(env_i)(v)$ unchanged. Therefore, for every $prefix \in Prefix$, every IP ip covered by $prefix$, and every router $v \in V$, the propagation path of $\mathcal{S}'(env)(v)(prefix)$ equals that of $\mathcal{S}'(env_i)(v)(prefix)$, and we have $\mathsf{fwd}(env, v, ip) = \mathsf{fwd}(env_i, v, ip)$. By applying the same reasoning to all messages in $SM_{APC_j^i}$ and all external neighbors in V_E, we can construct a $\widehat{env} \in \widehat{ENV}_{APC_j^i}$ that satisfies the above conditions.

Since $\widehat{ENV}_{APC_j^i} \subseteq ENV$, the above conditions also follow conversely. Thus, the network behaviors under $\widehat{ENV}_{APC_j^i}$ and ENV are equivalent. Applying the same reasoning to each APC_x^i associated with every $\widehat{attr}_{(i,x)} \in \widehat{Attr}_i$, we conclude that the network behaviors under ENV and $\widehat{ENV}_{Attr_i} = \mathcal{L} \times \widehat{\mathcal{N}}_{Attr_i}$ are also equivalent, where $\widehat{\mathcal{N}}_{Attr_i} = \{\widehat{n} | \widehat{n} : V_E \to 2^{Prefix \times Attr_1 \times \ldots \times \widehat{Attr_i} \times \ldots \times Attr_n}\}$. □

Lemma 2. *Let h_p be an effective prefix reduction function. The network behaviors under the full environment space ENV and the space $\widehat{ENV}_{pfx}$ reduced by h_p are equivalent.*

Proof. Let $Prefix_o$ denote the set of prefixes originating from the internal routers, and let $Prefix_a$ denote the set of prefixes involved in route aggregation. We have that $\widehat{ENV}_{pfx} = \mathcal{L} \times \widehat{\mathcal{N}}_{pfx}$, where $\widehat{\mathcal{N}}_{pfx} = \{\widehat{n} | \widehat{n} : V_E \to 2^{\widehat{Prefix} \times Attr_1 \times \ldots \times Attr_n}\}$.

We begin by proving the equivalence of routing behavior: for every $prefix \in Prefix$ and every $env \in ENV$, there exists $\widehat{env} \in \widehat{ENV}_{pfx}$ such that, for every router $v \in V$, the propagation path of $\mathcal{S}'(env)(v)(prefix)$ equals that of $\mathcal{S}'(\widehat{env})(v)(h_p(prefix))$.

- **Case 1:** $prefix \in Prefix_o \cup Prefix_a$. Since h_p is both *originating effective* and *aggregating effective*, we have $(Prefix_o \cup Prefix_a) \subseteq \widehat{Prefix}$ and $h_p(prefix) = prefix$. Let $SM(v_e)$ denote the set of routing messages with prefixes in $Prefix_o \cup Prefix_a$ that external neighbor $v_e \in V_E$ sends under environment env. We construct $\widehat{env} \in \widehat{ENV}_{pfx}$ such that each v_e sends exactly $SM(v_e)$. Because the propagation and selection of routing messages with different prefixes are independent–except when dependencies are introduced via route aggregation– the behavior of messages with prefix $prefix$ is influenced solely by those in $SM(v_e)$. Messages outside $SM(v_e)$ do not affect the propagation and selection of $\mathcal{S}'(env)(v)(prefix)$ for every v. Consequently, for every $v \in V$, we have that the propagation path of $\mathcal{S}'(env)(v)(prefix)$ equals that of $\mathcal{S}'(\widehat{env})(v)(prefix)$.
- **Case 2:** $prefix \notin Prefix_o \cup Prefix_a$. Let $SM(v_e)$ denote the set of routing messages with prefix $prefix$ sent by $v_e \in V_E$ under env. We construct an environment $\widehat{env} \in \widehat{ENV}_{pfx}$, where each v_e sends a set $\widehat{SM}(v_e)$ such that each message in $\widehat{SM}(v_e)$ is identical to its counterpart in $SM(v_e)$ except that the prefix is replaced by $h_p(prefix)$. Since $prefix, h_p(prefix) \notin Prefix_a$, the propagation and selection of messages with either prefix are only influenced by messages with the same respective prefix. Furthermore, because $prefix, h_p(prefix) \notin Prefix_a$, all relevant routing messages originate from external neighbors. Given that h_p is *policy effective*, we can conclude that, for every router $v \in V$, $\mathcal{S}'(env)(v)(prefix)$ and $\mathcal{S}'(\widehat{env})(v)(h_p(prefix))$ differ only in the prefix field. Therefore, the propagation paths of these two messages are identical.

Next, we prove the equivalence of forwarding behavior: for every IP ip covered by $prefix$ and every $env \in ENV$, there exists $\widehat{ip}$ covered by $h_p(prefix)$ and an environment $\widehat{env} \in \widehat{ENV}_{pfx}$ such that $\mathsf{fwd}(env, v, ip) = \mathsf{fwd}(\widehat{env}, v, \widehat{ip})$ for every router $v \in V$. Let PFX denote the set of prefixes carried by the routing messages selected across the network for forwarding traffic to ip under env. Let $\widehat{PFX}$ be the set of prefixes mapped from PFX by h_p. Note that $prefix \in PFX$ and $h_p(prefix) \in \widehat{PFX}$. For each external neighbor $v_e \in V_E$, let $SM(v_e)$ be the set of routing messages sent by v_e whose prefix belongs to PFX. Since ip is covered by all prefixes in PFX, and different prefixes with the same mask length do not overlap, any two $prefix_x, prefix_y \in PFX$ must have a sub-prefix relationship. Since h_p is *sub-prefix effective*, $h_p(prefix_x)$ and $h_p(prefix_y)$ must also have a sub-prefix relationship. Now consider a corresponding environment $\widehat{env} \in \widehat{ENV}_{pfx}$. In $\widehat{env}$, each external neighbor v_e sends the same messages as in $SM(v_e)$, except that every prefix $prefix_x \in PFX$ in those messages is replaced with $h_p(prefix_x)$. From the previous proof, the propagation paths of $\mathcal{S}'(env)(v)(prefix_x)$ and $\mathcal{S}'(\widehat{env})(v)(h_p(prefix_x))$ are identical for all $v \in V$ and $prefix_x \in PFX$. Moreover, by the policy effectiveness of h_p, their next hops are also the same. As all prefixes in $\widehat{PFX}$ must cover a common IP $\widehat{ip}$, and longest-prefix matching is preserved under h_p, we can conclude that for every $v \in V$, $\mathcal{S}'(\widehat{env})(v)(h_p(prefix_x))$ will be selected for forwarding traffic to $\widehat{ip}$ whenever $\mathcal{S}'(env)(v)(prefix_x)$ is selected for ip. Thus, $\mathsf{fwd}(env, v, ip) = \mathsf{fwd}(\widehat{env}, v, \widehat{ip})$.

Since $\widehat{ENV}_{pfx} \subseteq ENV$, the above conditions also hold conversely. Therefore, Lemma 2 holds. □

Theorem 1. *Given an effective reduction function h and corresponding environment spaces ENV and $\widehat{ENV}$, the network behaviors under ENV and $\widehat{ENV}$ are equivalent.*

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

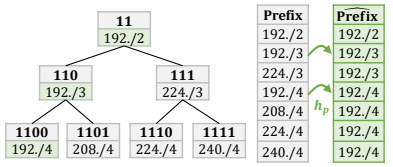


Fig. 4: Prefix trie and reduction over 192.0.0.0/2 (length 2-4).

Proof. Theorem 1 follows directly from Lemmas 1 and 2. We omit the proof due to space limitations. □

C. Reduction algorithm

We now present an algorithm for computing effective environment reductions directly from router configurations. The key challenge lies in ensuring that the derived prefix and attribute reduction functions satisfy the effective conditions defined in Section IV-A. Therefore, our focus in the following is on identifying such effective reduction functions.

Compute the effective prefix reduction function h_p. Since prefixes remain unchanged during message propagation, it suffices to analyze each transfer function trans independently, instead of considering the sequence of trans formed along the message propagation path.

Bulwark computes h_p as follows: First, Bulwark extracts a collection $\mathcal{P}$ of prefix sets from configurations. Each set $set \in \mathcal{P}$ corresponds to either (i) a group of prefixes matched by a single statement in each trans, or (ii) a singleton prefix that originates internally or involves route aggregation. To ensure completeness, the full prefix set $Prefix$ is also added to $\mathcal{P}$. Next, Bulwark refines $\mathcal{P}$ into a collection of disjoint sets, forming a partition over the prefix space $Prefix$. For each $set \in \mathcal{P}$, Bulwark constructs prefix tries over all prefixes in the set. It then selects the longest root-to-leaf path in each trie and collects all prefixes along this path. Finally, for each prefix $prefix$ on the path, h_p maps all prefixes in the trie with the same mask length as $prefix$ to $prefix$. Fig. 4 illustrates a prefix trie and the resulting reduction function.

Compute the effective attribute reduction functions h_{a_i}. All external routing messages exchanged between ASes are eBGP messages. Therefore, we only need to focus on BGP attribute reduction. As summarized in Table I, only a subset of BGP attributes [38]–*next hop*, *AS path*, *origin*, *MED*, and *community list*–are transitive across ASes. The remaining attributes are either assigned fixed values in eBGP messages or are reset upon AS traversal, and thus require no reduction.

Among the transitive attributes, some can also be omitted. The *next hop* attribute is deterministically set per neighbor and thus irrelevant to reduction. The *origin* has only three values (i.e., IGP, EGP, INCOMPLETE), with EGP being usually deprecated, making its reduction trivial. The *MED* attribute only affects message selection when comparing eBGP routing messages from the same AS. It serves as a preference hint from the external AS and is unrelated to internal properties that concern internal operators. Thus, its reduction is unnecessary.

Consequently, we mainly focus on the *community list* and *AS path* attributes. As shown in Algorithm 1, Bulwark first computes all possible propagation paths that each external routing message may traverse from every external neighbor by traversing the topology (lines 7-9). It then filters these paths to retain only those involving transfer functions that operate on community or AS paths. For each such path, it extracts the partial order of relevant transfer functions, which captures the order in which the community and AS path values are evaluated or transformed (lines 10-18). Then, each such transfer order is treated as a symbolic execution trace. For each trace, Bulwark computes a partition of attribute values such that all values within the same class are treated identically along the trace (lines 19–24). These classes are then refined into disjoint sets (lines 25-26), ensuring all values within the same class are treated identically along all traces. Note that each transfer function may have multiple match-action statements. Finally, Bulwark selects a representative value from each disjoint set and maps all values in the set to this representative (lines 27-35), forming $h_{a_{com}}$ and $h_{a_{asp}}$. To preserve *priority effectiveness*, Bulwark ensures that, when refining AS path equivalence classes (line 26), all AS paths within each set have identical length. It is worth noting that the computation can also incorporate complex policy operations like ADD-PATH and regex matching.

TABLE I: BGP attributes and their transitivity across ASes

BGP Attribute	Transitive Between ASes
Next hop, AS path, Origin, Community list Multi-exit discriminator (MED)	Yes
Local preference, Originator ID, Cluster list	No

Algorithm 1 Computing effective attribute reduction

Input: Network N, community attribute $Attr_{com}$, and AS path attribute $Attr_{asp}$
Output: Community reduction function $h_{a_{com}}$ and AS path reduction function $h_{a_{asp}}$

1: **procedure** ComputeEffectiveAttrReduction(N, $Attr_{com}$, $Attr_{asp}$)
2: $AllPaths \leftarrow \emptyset$
3: $OrderSet_{com} \leftarrow \emptyset$ ▷ All observed community policy orders
4: $OrderSet_{asp} \leftarrow \emptyset$ ▷ All observed AS path policy orders
5: $\mathcal{A}_{com}, \mathcal{A}_{asp} \leftarrow \emptyset$ ▷ Partitions for community and AS path values
6: $h_{a_{com}}, h_{a_{asp}} \leftarrow$ empty maps
7: **for** $v_e \in V_E$ **do**
8: $paths \leftarrow$ FindAllPaths(G, v_e)
9: $AllPaths \leftarrow AllPaths \cup paths$
10: **for** $path \in AllPaths$ **do**
11: $comOrder \leftarrow []$, $aspOrder \leftarrow []$
12: **for** $e \in path$ **do**
13: **if** HasCommunityStatement(trans$_e$) **then**
14: Append($comOrder$, trans$_e$)
15: **if** HasASPathStatement(trans$_e$) **then**
16: Append($aspOrder$, trans$_e$)
17: Add($OrderSet_{com}$, $comOrder$)
18: Add($OrderSet_{asp}$, $aspOrder$)
19: **for** $policyOrder \in OrderSet_{com}$ **do**
20: $\mathcal{A}_{com}^{policyOrder} \leftarrow$ ComputePartition($policyOrder, Attr_{com}$)
21: $\mathcal{A}_{com} \leftarrow \mathcal{A}_{com} \cup \mathcal{A}_{com}^{policyOrder}$
22: **for** $policyOrder \in OrderSet_{asp}$ **do**
23: $\mathcal{A}_{asp}^{policyOrder} \leftarrow$ ComputePartition($policyOrder, Attr_{asp}$)
24: $\mathcal{A}_{asp} \leftarrow \mathcal{A}_{asp} \cup \mathcal{A}_{asp}^{policyOrder}$
25: $\mathcal{A}_{com} \leftarrow$ DisjointifyPartitionClasses($\mathcal{A}_{com}$)
26: $\mathcal{A}_{asp} \leftarrow$ DisjointifyPartitionClasses($\mathcal{A}_{asp}$)
27: **for** $set \in \mathcal{A}_{com}$ **do**
28: $rep \leftarrow$ SelectRepresentative(set)
29: **for** $comm \in set$ **do**
30: $h_{a_{com}}(comm) \leftarrow rep$
31: **for** $set \in \mathcal{A}_{asp}$ **do**
32: $rep \leftarrow$ SelectRepresentative(set)
33: **for** $asp \in set$ **do**
34: $h_{a_{asp}}(asp) \leftarrow rep$
35: **return** $h_{a_{com}}, h_{a_{asp}}$
36: **end procedure**

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

Symbolic extension of the network instance:

$$
\begin{array}{lcl}
\mathcal{R}_E & = & \{R_e | R_e = \mathsf{True/False}, \forall e \in E\} \\
\mathcal{R}_M & = & \{R_m | R_m = \mathsf{True/False}, \forall m \in \widehat{n}_{all}(v_e), v_e \in V_E\} \\
EXPR & = & \{expr | expr \in \mathcal{G} = (\mathcal{R}_E, \mathcal{R}_M, \wedge, \vee, \mathsf{True}, \mathsf{False})\} \\
M^S & = & M \times EXPR \\
\mathsf{trans}^S & : & E \times M^S \to M^S \\
\oplus^S & : & M^S \times M^S \to 2^{M^S}
\end{array}
$$

Symbolic network solution: $\boxed{\mathcal{S}_S : V \to 2^{M_S}}$

$$
\mathcal{S}_S^t(v) = \begin{cases} \{(m, \mathsf{True}) | m \in \mathsf{originate(v)}\} & t{=}0, v{\in}V_I \\ \{(m, R_m) | m \in \widehat{n}_{all}(v), R_m \in \mathcal{R}_M\} & t{=}0, v{\in}V_E \\ \mathcal{S}_S^{t-1}(v) & t{>}0, v{\notin}\tau(t) \\ \mathcal{S}_S^0(v) \oplus^S (\bigoplus_{e=(u,v)\in E}^S \mathsf{trans}_e^S(\mathcal{S}_S^{t-1}(u)) & t{>}0, v{\in}\tau(t) \end{cases}
$$

Fig. 5: Symbolic extension of the network model.

D. Preserved network properties

Theorem 1 establishes that many network properties yield identical verification results under ENV and $\widehat{ENV}$:

- **Routing properties:** For any property related to the propagation or selection of routing messages with a prefix $prefix \in Prefix$ or an attribute value $attr \in Attr_i$, its verification result under ENV is identical to that under $\widehat{ENV}$ when verifying over the corresponding values $h_p(prefix) \in \widehat{Prefix}$ or $h_{a_i}(attr) \in \widehat{Attr_i}$.
- **Forwarding properties:** For any property related to the forwarding behavior of traffic destined to IP addresses covered by a prefix $prefix \in Prefix$, the verification result under ENV is identical to that under $\widehat{ENV}$, where destination IPs are mapped to those covered by the prefix $h_p(prefix) \in \widehat{Prefix}$.

V. Verification over the reduced environments

We have shown how to compute an effective reduction of the full environment space but do not illustrate how to efficiently explore this reduced space. We then describe how to extend the network model into a symbolic form, enabling scalable verification over the reduced environment space. As shown in Fig. 5, the network instance is extended with:

- $\mathcal{R}_E$ and $\mathcal{R}_M$ denote sets of symbolic boolean variables. Each variable $R_e \in \mathcal{R}_E$ (True/False) indicates the state (up/down) of link $e \in E$. Each variable $R_m \in \mathcal{R}_M$ (True/False) encodes whether an external neighbor $v_e \in V_E$ advertises a routing message m, where $m \in \widehat{n}_{all}(v_e)$, $v_e \in V_E$, and $\widehat{n}_{all} \in \widehat{\mathcal{N}}$. The $\widehat{n}_{all}$ maps each external neighbor to the full set defined by $\widehat{Prefix} \times \widehat{Attr_1} \times ... \times \widehat{Attr_n}$.
- $EXPR$ is the set of environment conditions, where each expression is a logical formula $expr$ from the boolean algebra $\mathcal{G}$ representing specific link states and external advertisement states.
- M^S is the set of symbolic routing messages, where each $m^s(m, expr)$ associates a message $m \in M$ with an environment condition $expr \in EXPR$ representing environments under which m is selected.
- trans^S is the symbolic extension of the transfer function trans. For a link $e = (u, v)$ and a symbolic message $m^s(m, expr)$, the $\mathsf{trans}_e^S(m^s)$ extends trans_e by updating the environment condition of m^s to $expr \wedge R_e$, indicating that m can propagate through e only when the link is up.
- $\oplus^S$ is the symbolic extension of $\oplus$, modeling the selection process of symbolic routing messages. Unlike $\oplus$, $\oplus^S$ preserves all messages with different preferences by restricting their environment conditions. For $m_1^s = (m_1, expr_1)$ and $m_2^s = (m_2, expr_2)$ in M^S, the result of $m_1^s \oplus m_2^s$ is defined as follows: (1) If m_1 and m_2 have different prefixes, both messages are retained: $\{m_1^s, m_2^s\}$. (2) If m_1 and m_2 share the same prefix, the result is $\{m_1^s - m_2^s,\ m_2^s - m_1^s\}$, where $m_1^s - m_2^s$ is computed based on their preference: (i) If m_1 is preferred over m_2, then $m_1^s - m_2^s = (m_1,\ expr_1)$, indicating that m_1 is always selected when available. (ii) If m_1 is less preferred than m_2, then $m_1^s - m_2^s = (m_1,\ expr_1 \wedge \neg expr_2)$, meaning m_1 is selected only when m_1 is available and m_2 is not.

Given a symbolic network instance, its symbolic solution $\mathcal{S}_S$ maps each router vertex $v \in V$ to a set of symbolic routing messages that may be selected by v under different environment conditions. At time step $t = 0$, the solution $\mathcal{S}_S^0(v)$ for each internal router $v_i \in V_I$ is initialized with all messages originating from v_i, each associated with the condition True, indicating that these messages are always available to v_i regardless of the environment. For each external neighbor $v_e \in V_E$, $\mathcal{S}_S^0(v_e)$ includes all messages in $\widehat{n}_{all}(v_e)$, each paired with a symbolic variable R_m, which encodes that message m is available only if it is actually advertised by v_e. For subsequent time steps ($t > 0$), the set of symbolic messages selected by each router evolves according to the symbolic transfer function and message merging semantics. Fig. 6 shows an example of symbolic simulation.

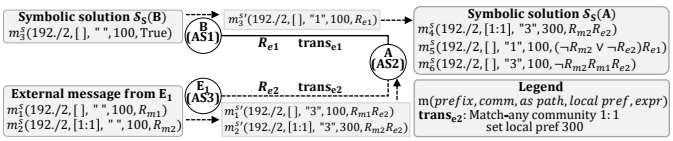


Fig. 6: An illustrative example of symbolic simulation.

VI. Evaluation

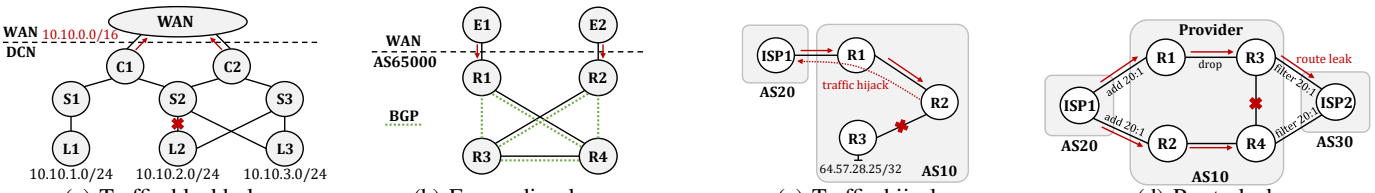
We evaluate Bulwark to demonstrate: (i) its effectiveness in detecting both known and novel configuration bugs in synthetic and real-world scenarios, and (ii) its performance across networks of varying sizes. Bulwark is implemented in Java using Binary Decision Diagrams (BDDs), with the JDD library [39]. For prefix reduction, we select one representative prefix per trie; for AS path partitioning, we consider only the shortest AS paths. These heuristics are useful and introduce no false positives or negatives in our experiments. To enhance scalability, we apply several optimizations like [15], [16], [40], including symbolic variable reduction, routing message pruning, and prefix pruning.

All evaluations ran on a Linux server with an AMD EPYC 7402 24-Core Processor and 128 GB of memory.

A. Finding bugs in synthetic scenarios with Bulwark

We demonstrate Bulwark's ability to detect both known and novel bugs in synthetic scenarios, which are derived from

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.



(a) Traffic blackhole (b) Forwarding loop (c) Traffic hijack (d) Route leak

Fig. 7: Known and new bugs in synthetic scenarios.

TABLE II: Statistics of the dataset.

Dataset	Routers	Links	Neighbors	Policies	Config-Lines
Internet2	∼10	∼80	∼300	∼3k	∼100k
Campus	∼10	∼20	∼10	∼30	∼2k
Net1	∼10	∼10	∼10	∼0	∼1k
US Carrier	∼150	∼200	∼30	∼600	∼20k
Columbus	∼70	∼90	∼20	∼350	∼10k
Bics	∼30	∼50	∼10	∼200	∼5k

TABLE III: Known and new bugs found by Bulwark.

Network	Prefix Hijack		Route Leak		NoMartian		BTE		Blackhole	
	Known	New	Known	New	Known	New	Known	New	Known	New
Campus	2	8	0	2	-	-	-	-	-	-
Net1	0	3	0	1	-	-	-	-	2	0
Internet2	0	95	-	-	0	1	5	0	-	-

TABLE IV: Runtime on Internet2 (Bagpipe vs. Bulwark).

Property	Bagpipe	Bulwark
NoMartian	1,178s (20min)	113s
BTE	28,594s (8h)	115s
PrefixHijack	–	117s

TABLE V: External advertisement state reduction ratio.

Network	$\mathcal{N}/\widehat{\mathcal{N}}$
Bics	$> 10^{4.27\times10^{11}}$
Columbus	$> 10^{8.53\times10^{11}}$
US Carrier	$> 10^{1.28\times10^{12}}$

real-world incidents. Due to a lack of full configurations, we replicate these cases. All bugs are latent–triggered only under specific environments–and even in small-scale topologies, existing tools either miss them or fail to scale effectively.

1) Finding known bugs in synthetic scenarios:

- **Traffic blackhole due to erroneous route aggregation.** In the scenario illustrated in Fig.7a (adapted from [18]), traffic destined for prefix 10.10.2.0/24 from the WAN may be blackholed at router $C1$ upon failure of the $S2$-$L2$ link. The data center network (DCN) hosts services associated with prefixes 10.10.1.0/24, 10.10.2.0/24, and 10.10.3.0/24 on leaf routers $L1$, $L2$, and $L3$, respectively. To enable WAN connectivity, core routers $C1$ and $C2$ aggregate these prefixes into a single advertisement for 10.10.0.0/16 and announce it to the WAN. However, due to incorrect aggregation configuration, $C1$ continues to advertise the aggregate even after losing reachability to $L2$. Consequently, traffic to 10.10.2.0/24 is forwarded to $C1$, which lacks a more specific route and therefore drops the packets, resulting in a forwarding blackhole.
- **Forwarding loop caused by error preferences.** As shown in Fig. 7b (from [41]), a forwarding loop forms between routers $R3$ and $R4$ when $R1$ and $R2$ receive external messages with the same prefix. The network runs BGP, with $R3$ and $R4$ serving as route reflectors for $R1$ and $R2$. Due to misconfiguration, $R3$ prefers $R1$ as the exit, while $R4$ prefers $R2$. As a result, $R3$ forwards traffic to $R4$, and $R4$ sends it back to $R3$, forming a loop.

2) Finding new bugs in synthetic scenarios:

- **Prefix hijacking due to poor routing policies.** As shown in Fig. 7c, traffic to the internal IP address 64.57.28.25 can be hijacked by ISP1 if the $R2$-$R3$ link fails and ISP1 advertises a spoofed route. The service resides on $R3$, with ISIS for internal routing and BGP for external routes, where $R2$ serves as a route reflector. Normally, ISIS routes take precedence, keeping traffic internal. However, an error import policy on $R1$ fails to filter internal prefixes, allowing it to accept the bogus BGP route when ISIS is unavailable and divert traffic to ISP1.
- **Route leak due to error BGP neighbor configuration.** As illustrated in Fig. 7d, a route leak from ISP1 to ISP2 occurs when the $R3$-$R4$ link fails. The network uses BGP, with $R3$ and $R4$ as route reflectors. To prevent leaks, $R1$ and $R2$ tag messages from ISP1 with community 20:1, and $R3$ and $R4$ apply export filters that drop such messages before reaching ISP2. Without link failures, $R3$ prefers ISP1 routes via $R2$, ensuring proper filtering. However, since the BGP session between $R1$ and $R3$ is configured without send – community, community tags are omitted. If the $R3$-$R4$ link is down, $R3$ may select untagged routes from $R1$ and propagate them to ISP2, bypassing filters and triggering a leak.

B. Finding bugs in real scenarios with Bulwark

We further evaluate Bulwark on three real-world networks: a campus network [7], a Batfish-analyzed network (Net1) [6], and the nationwide WAN Internet2 [24] (Table II). As shown in Table III, Bulwark detects all previously reported bugs and uncovers several new misconfigurations missed by prior tools.

1) Finding known bugs in real scenarios:

- **Prefix Hijack.** In the campus network, services under prefix 2.128.0.0/24 must be reachable by all internal routers. However, Bulwark identified that traffic is incorrectly redirected to AS1 when AS2's border router receives external routes from AS1 containing sub-prefixes of 2.128.0.0/24 and communities matching '1:*'. This misbehavior is caused by an incorrect import policy that fails to filter routes for 2.128.0.0/24 and its sub-prefixes.
- **No Martian.** The noMartian property requires routers to discard routes with 'martian' prefixes (e.g., 10.0.0.0/8), which should not appear on the public Internet. Internet2 enforces this property, but Bulwark detected that a router could select a sub-prefix of a martian prefix. Further analysis suggests this may be due to intentional configuration.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

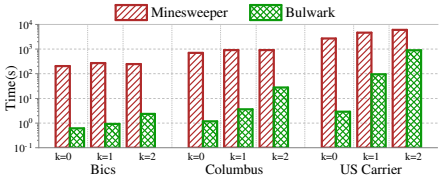


Fig. 8: Runtime across WANs under all environments.

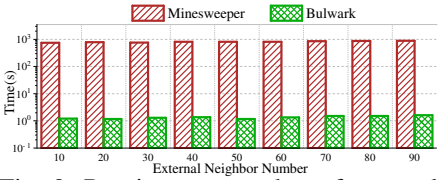


Fig. 9: Runtime vs. number of external neighbors.

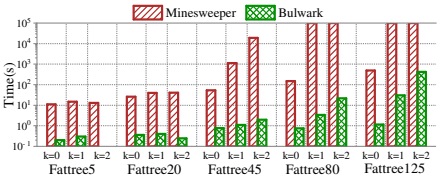


Fig. 10: Runtime across fat trees under k link failures.

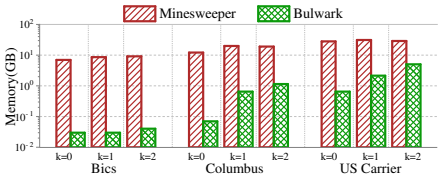


Fig. 11: Memory usage across WANs under all environments.

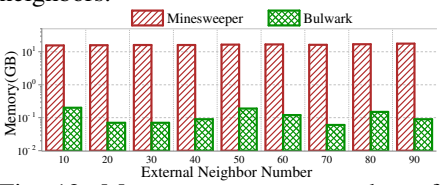


Fig. 12: Memory usage vs. number of external neighbors.

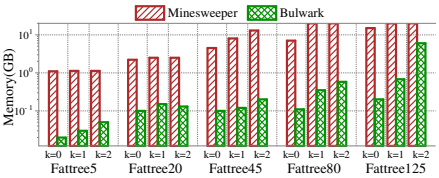


Fig. 13: Memory usage across fat trees under k link failures.

- **Block To External.** The BlockToExternal property ensures that routes tagged with communities like NoExport (e.g., 11537:888) are not propagated to external neighbors. Internet2 enforces this policy via the SANITY-OUT filtering policy. Bulwark detected violations involving five border routers, where SANITY-OUT was not applied, allowing export of messages with BTE communities.

2) Finding new bugs in real scenarios:

Campus network. Bulwark uncovered eight potential prefix hijacks within AS1 and AS3 of the campus network. In AS1, which hosts 1.0.1.0/24, traffic should remain internal. However, when border routers receive external messages from AS2 with sub-prefixes of 1.0.1.0/24 and communities matching '2:*', traffic is redirected to AS2. This was caused by the error import policies that fail to filter these sub-prefixes.

Bulwark also detected two possible route leaks. While AS1 and AS3 should independently access AS2 and avoid providing transit for one another, AS1 was found to leak routes–received from AS2–to AS3, potentially enabling free transit. A similar misconfiguration was identified in AS3.

Net1. We identified three new prefix hijacking vulnerabilities in Net1. The prefix 10.0.0.0/24, hosted on router $N2$ in AS2, is internally advertised via OSPF and should receive traffic destined to it. Bulwark revealed that when AS1 or AS3 advertises sub-prefixes of 10.0.0.0/24 to AS2, traffic is misrouted to the external AS instead of $N2$. The root cause is the lack of sub-prefix filtering on AS2's border routers.

Internet2. In Internet2, eight internal prefixes are reserved for backbone services, and traffic to these prefixes should remain within the network. Bulwark found that 75 external neighbors could hijack traffic to seven of them by redirecting it externally. The root cause is that Internet2's import policies filter only exact matches, allowing sub-prefixes to bypass. For the remaining prefix, 20 neighbors could still hijack traffic due to insufficient sub-prefix filtering, despite stricter policies.

C. Verification performance

1) Performance on real networks: We compare the performance of Bulwark and Bagpipe [8] on Internet2 for verifying three properties: NoMartian, BlockToExternal, and Prefix Hijack (only supported by Bulwark). Since Bagpipe is not open-source, we use the results reported in its paper. As shown in Table IV, Bulwark verifies all properties in around 100 seconds, whereas Bagpipe takes over 8 hours in the worst case. Bulwark is about $10\times$ faster for NoMartian and nearly $100\times$ faster for BlockToExternal.

2) Performance on synthetic networks: Table V shows the reduction ratio of external advertisement states on three synthetic networks. Bulwark reduces the environment space by over $10^{10^{11}}\times$ on all networks.

We further compare Bulwark with Minesweeper, the only verifier that supports verification under all environments. We use three networks of increasing size–Bics, Columbus, and US Carrier–from [40]. Each includes a different number of routers, links, routing policies, and external neighbors. We focus on verifying the traffic hijack property under environments with up to two link failures ($k < 3$) and all external advertisement states. The property is detected by checking pairwise reachability. As shown in Fig. 8, Bulwark outperforms Minesweeper by nearly two orders of magnitude for $k = 0$ and $k = 1$, and by nearly $10\times$ for $k = 2$. While runtime grows with k, failures beyond $k = 2$ are rare in the real world [42].

We also compare Bulwark and Minesweeper on the Columbus networks with varying numbers of external neighbors. As shown in Fig. 9, Bulwark's runtime remains nearly constant, consistently $100\times$ faster than Minesweeper. To further evaluate scalability, we evaluate Bulwark on fat-tree networks with 5 to 500 links (Fattree5–Fattree125) under environments with k link failures. As shown in Fig. 10, Bulwark is nearly $100\times$ faster than Minesweeper across all scales.

Finally, we compare memory usage. As shown in Figures 11, 12, and 13, Bulwark's peak memory usage stays below 10 GB, while Minesweeper exceeds 20 GB.

VII. Conclusion

We introduce Bulwark, a network configuration verifier that effectively reduces the full environment space to a small and sound subset, enabling scalable verification under all possible environments. Bulwark is based on a theory of network behavior equivalence between the full environment space and its effective reduction. Experimental results show that Bulwark detects many known and previously unknown configuration bugs in real-world and synthetic networks and outperforms existing configuration verifiers by up to $100\times$ in speed.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] T. Anderson, "Ovh blames hour-long global outage on human error during 'routine' network reconfiguration," 2021. [Online]. Available: https://www.theregister.com/2021/10/13/ovh_outage/

[2] Wikipedia, "2021 facebook outage," 2021. [Online]. Available: https://en.wikipedia.org/wiki/2021_Facebook_outage

[3] K. Mackie, "Configuration glitch caused microsoft's jan. 25 exchange online disruption," 2023. [Online]. Available: https://redmondmag.com/articles/2023/01/26/configuration-glitch-caused-microsoft-jan-25-exchange-online-disruption.aspx

[4] J. H. Tom Strickx, "Cloudflare outage on june 21, 2022," 2022. [Online]. Available: https://blog.cloudflare.com/cloudflare-outage-on-june-21-2022/

[5] A. Siddiqui, "Bgp route leak at angola cables slows connectivity for many australians," 2023. [Online]. Available: https://manrs.org/2023/05/bgp-route-leak-at-angola-cables-slows-connectivity-for-many-australians/

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

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

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

[9] T. Alberdingk Thijm, R. Beckett, A. Gupta, and D. Walker, "Modular control plane verification via temporal invariants," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 50–75, 2023.

[10] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Control plane compression," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 476–489.

[11] ——, "Abstract interpretation of distributed network control planes," *Proceedings of the ACM on Programming Languages*, vol. 4, no. POPL, pp. 1–27, 2019.

[12] A. Tang, R. Beckett, S. Benaloh, K. Jayaraman, T. Patil, T. Millstein, and G. Varghese, "Lightyear: Using modularity to scale bgp control plane verification," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 94–107.

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

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

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

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

[17] D. Wang, P. Zhang, and A. Gember-Jacobson, "Expresso: Comprehensively reasoning about external routes using symbolic simulation," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 197–212.

[18] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese, "Efficient network reachability analysis using a succinct control plane representation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 217–232.

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

[20] N. Giannarakis, D. Loehr, R. Beckett, and D. Walker, "Nv: An intermediate language for verification of network control planes," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 958–973.

[21] T. A. Thijm, R. Beckett, A. Gupta, and D. Walker, "Kirigami, the verifiable art of network cutting," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2447–2462, 2024.

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

[23] N. Giannarakis, A. Silva, and D. Walker, "Probnv: probabilistic verification of network control planes," *Proceedings of the ACM on Programming Languages*, vol. 5, no. ICFP, pp. 1–30, 2021.

[24] "Internet2." [Online]. Available: https://vn.net.internet2.edu/Internet2/

[25] "Z3." [Online]. Available: https://github.com/Z3Prover/z3

[26] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 113–126.

[27] R. Beckett and A. Gupta, "Katra: Realtime verification for multilayer networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 617–634.

[28] X. Liu, P. Zhang, H. Li, and W. Sun, "Modular data plane verification for compositional networks," *Proceedings of the ACM on Networking*, vol. 1, no. CoNEXT3, pp. 1–22, 2023.

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

[30] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese, "Checking beliefs in dynamic networks," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 499–512.

[31] Q. Xiang, C. Huang, R. Wen, Y. Wang, X. Fan, Z. Liu, L. Kong, D. Duan, F. Le, and W. Sun, "Beyond a centralized verifier: Scaling data plane checking via distributed, on-device verification," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 152–166.

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

[33] A. Tang, S. K. R. Kakarla, R. Beckett, E. Zhai, M. Brown, T. Millstein, Y. Tamir, and G. Varghese, "Campion: debugging router configuration differences," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 748–761.

[34] S. K. R. Kakarla, F. Y. Yan, and R. Beckett, "Diffy: Data-driven bug finding for configurations," *Proceedings of the ACM on Programming Languages*, vol. 8, no. PLDI, pp. 199–222, 2024.

[35] S. K. R. Kakarla, A. Tang, R. Beckett, K. Jayaraman, T. Millstein, Y. Tamir, and G. Varghese, "Finding network misconfigurations by automatic template inference," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 999–1013.

[36] R. Mondal, A. Tang, R. Beckett, T. Millstein, and G. Varghese, "What do llms need to synthesize correct router configurations?" in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023, pp. 189–195.

[37] C. Wang, M. Scazzariello, A. Farshin, S. Ferlin, D. Kostić, and M. Chiesa, "Netconfeval: Can llms facilitate network configuration?" *Proceedings of the ACM on Networking*, vol. 2, no. CoNEXT2, pp. 1–25, 2024.

[38] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," Tech. Rep., 2006.

[39] A. Vahidi, "JDD." [Online]. Available: https://bitbucket.org/vahidi/jdd/src/master/

[40] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "{Config2Spec}: Mining network specifications from network configurations," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 969–984.

[41] F. Le, G. G. Xie, and H. Zhang, "Instability free routing: Beyond one protocol instance," in *Proceedings of the 2008 ACM CoNEXT Conference*, 2008, pp. 1–12.

[42] N. Jara, H. Pempelfort, G. Rubino, and R. Vallejos, "A fault-tolerance solution to any set of failure scenarios on dynamic wdm optical networks with wavelength continuity constraints," *IEEE Access*, vol. 8, pp. 21 291–21 301, 2020.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15,2026 at 12:53:22 UTC from IEEE Xplore. Restrictions apply.