Scalable and Interpretable Overlay Network Checking via Ensemble Verification

XINZHE LIU, Tsinghua University and Zhongguancun Laboratory, China
YAHUI LI*, Tsinghua University, China
HAN ZHANG, Tsinghua University and Zhongguancun Laboratory, China
XIA YIN, Tsinghua University and Zhongguancun Laboratory, China
XINGANG SHI, Tsinghua University and Zhongguancun Laboratory, China
ZHILIANG WANG, Tsinghua University and Zhongguancun Laboratory, China
GANG REN, Tsinghua University and Zhongguancun Laboratory, China
JILONG WANG, Tsinghua University and Zhongguancun Laboratory, China
JIANGYUAN YAO, Hainan University, China

Overlay/underlay architecture is increasingly prevalent in modern networks but also introduces greater complexity and error-proneness. Existing control plane verifiers for underlay networks face scalability and interpretability challenges when extended to overlay/underlay networks due to methodological limitations. This paper presents MEV, the first control plane verifier designed for overlay/underlay networks. MEV introduces ensemble verification, a novel verification paradigm enabling independent reasoning for behaviors within each routing instance or protocol and forwarding behaviors within each virtual network. We implement and deploy MEV on a real nationwide overlay/underlay network, FITI, where it successfully identifies 22 misconfigurations that could cause isolation and reachability issues. Furthermore, we evaluate MEV against Batfish$^+$ on FITI and a range of synthetic networks with diverse scales, and find that it achieves up to a $10^2\times$ speedup over Batfish$^+$.

CCS Concepts: • **Networks** → **Protocol testing and verification**.

Additional Key Words and Phrases: Network Verification, Overlay Network, Ensemble Verification

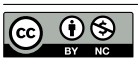
ACM Reference Format:
Xinzhe Liu, Yahui Li, Han Zhang, Xia Yin, Xingang Shi, Zhiliang Wang, Gang Ren, Jilong Wang, and Jiangyuan Yao. 2025. Scalable and Interpretable Overlay Network Checking via Ensemble Verification. *Proc. ACM Netw.* 3, CoNEXT4, Article 27 (December 2025), 25 pages. https://doi.org/10.1145/3768974

1 Introduction

With the widespread adoption of network design principles such as virtualization and multi-tenancy, the overlay/underlay architecture has become a foundational paradigm in modern data centers and cloud environments. In many scenarios, overlay networks are typically established using distributed routing protocols such as BGP EVPN [1] and BGP VPNv6 [2], which dynamically or

*Corresponding author.

Authors' Contact Information: Xinzhe Liu, lxz23@mails.tsinghua.edu.cn, Tsinghua University and Zhongguancun Laboratory, China; Yahui Li, liyahui@mail.tsinghua.edu.cn, Tsinghua University, China; Han Zhang, zhhan@tsinghua.edu.cn, Tsinghua University and Zhongguancun Laboratory, China; Xia Yin, Tsinghua University and Zhongguancun Laboratory, China; Xingang Shi, Tsinghua University and Zhongguancun Laboratory, China; Zhiliang Wang, Tsinghua University and Zhongguancun Laboratory, China; Gang Ren, Tsinghua University and Zhongguancun Laboratory, China; Jilong Wang, Tsinghua University and Zhongguancun Laboratory, China; Jiangyuan Yao, Hainan University, China.



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2834-5509/2025/12-ART27
https://doi.org/10.1145/3768974

statically construct Virtual Private Clouds (VPCs) or Virtual Private Networks (VPNs) atop existing ones. In the data plane, encapsulation protocols such as SRv6 [3], NVGRE [4], and IP GRE [5] are employed to form tunnels and encapsulate payloads. For instance, major cloud providers like Google and Microsoft [6, 7] leverage BGP EVPN L3VPN over VXLAN tunnels to isolate tenant traffic by virtualizing IP networks into logically independent overlay segments.

While the overlay/underlay architecture has been proven successful, enabling the development of new features and services atop existing infrastructure, it also increases network complexity and error-proneness. For example, in AT&T's backbone, packets are encapsulated up to eleven times across various virtual networks [8], and an outage in any one of them can disrupt all others relying on it. Given the ubiquity of overlay/underlay design, ensuring its reliability is crucial.

However, different from the underlay network, the overlay/underlay network proposes new challenges for network verifiers due to its unique characteristics:

- **More complex dependent conditions between routing instances or protocols in the control plane.** Dependent conditions in the control plane refer to the communication supports (e.g., HELLO or TCP message reachability) required to establish communication sessions between routing processes within the same routing instance or protocol, which are ensured by others. In underlay networks, dependent conditions between routing instances or protocols are typically simple, with BGP depending on IGP, and IGP not depending on anything else. However, in the overlay/underlay networks, routing processes can interact based on the support provided by virtual networks established by overlay processes and, in turn, create new virtual networks to provide support for others. This results in more complex and lengthy dependency chains in the control plane. For example, as shown in Figure 1a, the interaction between BGP EVPN processes on routers CE_3 and CE_4 depends on communication supports provided by virtual networks built by BGP EVPN processes in the provider network $N1$, which in turn depends on its IGP protocol. As network services grow more complex, these conditions may become more convoluted.
- **More complex dependent conditions between virtual networks in the data plane.** As new virtual networks are often built on top of existing ones, packet forwarding within a specific virtual network N_1 may depend on other virtual or physical networks N_2 for forwarding the encapsulated packets. This creates more complex dependent conditions in the data plane. For example, in the network shown in Figure 1a, packets from $VM3$ to $VM1$ are first encapsulated into a VXLAN tunnel and then further encapsulated into an SRv6 tunnel to reach the VXLAN tunnel endpoint, with each tunnel belonging to a distinct virtual network.
- **Harder interpretable errors.** The complex dependent conditions in overlay/underlay networks often cause errors to span across multiple routing instances or protocols, and virtual networks, affecting all network behaviors dependent on it. Consequently, localizing the root cause of the error becomes more challenging. For instance, as shown in Figure 1a, an error ACL on $port_1$ of PE_1 causes unreachability in the SRv6 tunnel, disrupting interactions between BGP EVPN processes on CE_3 and CE_4. This leads to unreachability between $VM3$ and $VM1$ (no routes to each other). However, with only the top unreachability symptom, it is difficult to narrow down the scope of the error to $N1$.

Although many control plane verifiers have been proposed for underlay networks, none are capable of verifying overlay/underlay networks with high scalability and interpretability due to the characteristics outlined above. Monolithic verifiers [9–18] couple the entire network into a single model based on dependent conditions within the control and data planes. They: (i) must iteratively compute the behaviors of dependent processes and the processes that depend on them, based on their dependent relationships; (ii) require complex packet encapsulations and redundant forwarding for encapsulated packets when multiple virtual networks share the same tunnel; and

(iii) can only identify errors based on their impact on the topmost section of the dependency chain when the error occurs at a specific link in the chain. Modular verifiers [19–21] perform local checks on each router but struggle with fully automating verification and reasoning about the forwarding behaviors within overlay/underlay networks modularly.

Therefore, this paper presents MEV, the first control plane verifier designed for overlay/underlay networks. MEV ensures highly scalable verification in networks with complex dependent conditions by: (i) enabling independent computation of control plane states for different routing instances or protocols, and (ii) allowing independent reasoning for forwarding behaviors within individual virtual or physical networks. Furthermore, MEV improves interpretability by providing the root cause and a causal chain for each error, illustrating how it spans across routing instances, protocols, and virtual or physical networks, ultimately leading to property violations.

To achieve this, MEV proposes a novel verification paradigm termed ensemble verification. In the control plane, MEV decouples dependent conditions between different routing instances or protocols into each single one, isolating them into disjoint process components. Within each component, routing messages are propagated with corresponding conditions, enabling the independent computation of control plane states. In the data plane, MEV decouples forwarding behaviors across virtual or physical networks N_1 and N_2 into two distinct parts: the forwarding of pre-encapsulated (decapsulated) packet in N_1 and the forwarding of encapsulated packet in N_2, and treats forwarding behaviors in N_2 as dependent conditions of N_1. Independent reasoning for forwarding behaviors is then performed within N_1 and N_2. Moreover, MEV regards the decoupling of forwarding behavior as an atomic operation. In the case of nested encapsulations, MEV iteratively decouples the forwarding behavior into each virtual or physical network for independent reasoning. Finally, MEV integrates and independently verifies all dependent conditions decoupled above, which is required for network behaviors specified by properties to be verified, completing the ensemble verification.

Leveraging ensemble verification, MEV clearly identifies all dependent conditions required for a property to hold and their interrelationships. By traversing the graph formed by these conditions, MEV identifies the root causes of property violations and constructs the causal chains from each root cause to the violated property, explaining how the root cause lead to the violation.

We implement MEV and deploy it on FITI [22], a nationwide overlay/underlay network, over a five-month period. During this deployment, MEV successfully detected 22 misconfigurations that would cause unreachability or isolation violations across virtual networks. We further evaluated MEV against Batfish$^+$–an extended version of Batfish [10], which is among the most popular and scalable control plane verifiers when arbitrary environments are not considered–on FITI and a range of synthetic networks spanning 5 to 502 routers, finding MEV achieves up to a $10^2\times$ speedup.

Our contributions with MEV are the following:

- We introduce a novel verification concept, ensemble verification, which supports verifying the overlay/underlay network with high scalability and interpretability.
- We present a new control plane model for overlay/underlay networks that supports independent computation of control plane states for each routing instance or protocol.
- We implement MEV and deploy it to a nationwide overlay/underlay network (FITI), showing its effectiveness in practical applications.
- We compare the performance of MEV with Batfish$^+$ on a large set of synthetic networks, illustrating its scalability and interpretability in networks with different scales.

2 Motivation

2.1 Verification requirements

The unique characteristics of overlay/underlay networks, as discussed in §1, impose two key requirements on network verifiers:

- **Scalability***: efficiently verifying network properties in the presence of complex dependent conditions within control and data planes.*
- **Iterpretability***: providing precise root causes and causal explanations for property violations, even when errors span routing instances or protocols, and virtual networks.*

2.2 Limitations of existing verifiers

Control plane verifiers analyze network configurations to reason about resulting behaviors, proactively preventing errors before deployment. However, none can extend to overlay/underlay networks while meeting the above requirements due to their methodological limitations:

(1) Monolithic verification. Many verifiers [9–18, 23–25] couple the control plane as a monolithic model based on the dependent conditions and interactions between different processes. In the monolithic model, they first analyze the behavior of dependent processes, then proceed to analyze the processes that depend on them (e.g., IGP before BGP in Batfish). This iterative procedure continues until the control plane state of the entire network converges. Similarly, when reasoning about packet forwarding behaviors, they forward packets across different virtual and physical networks monolithically. This means, for packets forwarded to the next hop through tunnels within a specific virtual network, they may perform the encapsulation operation, forward the encapsulated packet to the tunnel endpoint through other virtual or physical networks, and then continue with forwarding the decapsulated packet. If multiple packets go through the same tunnel, they forward the encapsulated packet repeatedly. All of these make them struggle with scalability issues when verifying overlay/underlay networks. Furthermore, in the monolithic verification, any error directly impacts all network behaviors that depend on it, causing the error to span across routing instances or protocols, and virtual networks. This means verifiers can only identify errors based on their manifestations at the top of the dependency chain. Therefore, it is difficult for them to localize the root cause and interpret the error. For example, in the error discussed in §1, they will only return the top-level symptom–no route to $VM1$–offering little assistance in locating the error to the provider network $N1$. Incremental verifiers [26, 27] adopt a similarly monolithic philosophy, albeit using different modeling approaches (e.g., Differential Datalog [28]) which enables them to incrementally compute control and data plane states in response to configuration updates. Therefore, they also face scalability issues (in the first-time verification) and interpretability issues.

(2) Modular verification. Many tools [19–21] break down the monolithic model into several small models for individual routers and perform local checks for each router independently. They focus on verifying routing properties (e.g., routing message reachability). During verification, they require users to specify the "interface" for each router, which is the assumption about the routing messages each router may receive from neighbors, making the verification not fully automatic. More importantly, the complex dependent conditions between different virtual networks make it difficult to extend to verify forwarding properties modularly, due to the difficulty of assuming what encapsulated packet each router may receive from neighbors. However, the forwarding operation is a main feature of the overlay/underlay network.

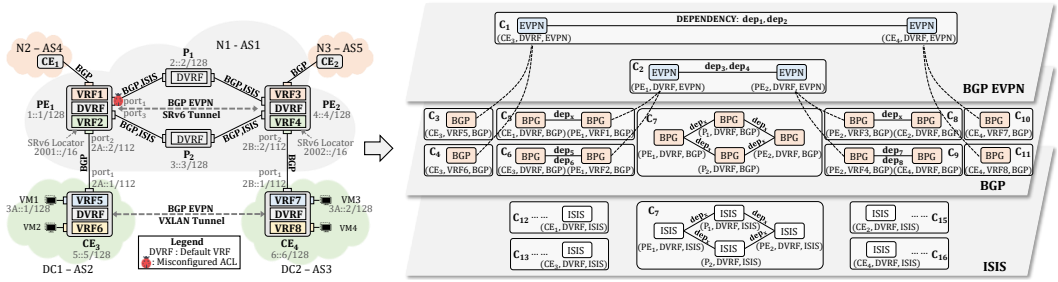
Data plane verifiers [29–36] operate in a space orthogonal to MEV and cannot detect misconfigurations prior to deployment. Static analysis tools–such as outlier detectors [37–39] and LLM-based methods [40, 41]–also face significant limitations, including the need for highly similar configurations across routers, limited accuracy, and inability to reason about network behavior.

3 Design overview of MEV

We then illustrate the key insights behind MEV and show the workflow of MEV with an example.

3.1 Key insight

Key insight 1: limited connections among routing instances or protocols. In the control plane, each routing process exchanges messages with peers within the same routing instance or protocol,



(a) A common scenario in cloud service providers (CSPs) and data centers (DCs).

Dep(Src router, Src VRF, Dst router, Dst VRF, Dst prefix)	
dep_1(CE$_3$, DVRF, CE$_4$, DVRF, 6::6/128)	dep_5(CE$_3$, DVRF, PE$_1$, VRF2,2A::2/128)
dep_2(CE$_4$, DVRF, CE$_3$, DVRF, 5::5/128)	dep_6(PE$_1$, VRF2, CE$_3$, DVRF, 2A::1/128)
dep_3(PE$_1$, DVRF, PE$_2$, DVRF, 4::4/128)	dep_7(CE$_4$, DVRF, PE$_2$, VRF4, 2B::2/128)
dep_4(PE$_2$, DVRF, PE$_1$, DVRF, 1::1/128)	dep_8(PE$_2$, VRF4, CE$_4$, DVRF, 2B::1/128)

(b) The dependent condition table for example network.

(c) Ensemble control plane model: a dependency process component C for each routing protocol or instance; a vertex represents a routing process (e.g., BGP) configured under a VRF (e.g., $VRF5$) on a router (e.g., CE_3); an edge $\rightarrow$ indicates routing message exchange within the same protocol or instance; an edge $\dashrightarrow$ represents the interaction between different protocols or instances; each edge $\rightarrow$ is associated with a set of dependent conditions dep, representing communication support required from other components.

Fig. 1. An example network and its ensemble control plane model. (a) $DC1$ and $DC2$ build virtual networks using EVPN L3VPN over VXLAN, ensuring connectivity between VMs of the same tenant (e.g., $VM1$, $VM2$). $N1$ serves as the DCI, using EVPN L3VPN over SRv6 to ensure connectivity between CEs (e.g., CE_3, CE_4).

selects the best one for each prefix, and propagates it to peers accordingly. In addition, inter-instance or inter-protocol communication occurs via route redistribution, governed by redistribution policies. This iterative procedure continues until the control plane states of all routing instances and protocols converge. Finally, each router combines the best routes selected by local processes to construct its RIB, and all routers' RIBs constitute the global control plane state.

During the above procedure, connections between routing instances and protocols fall into two categories: (1) dependent conditions with high complexity, as defined in §1; and (2) simple route redistribution relationships. Importantly, the content (e.g., reachability between peer IPs) of dependent conditions required for communication between processes within a specific instance or protocol is determined solely by their own configurations, independent of the supporting processes. Therefore, the entire control plane can be decoupled into a series of disjoint process components, each corresponding to a single instance or protocol. The conditions required for communication between processes are encoded on the edges between process vertices. The control plane state of each routing instance or protocol can be computed independently through propagating routing messages that carry all dependent conditions on the propagation path. Finally, the global control plane state is obtained by combining all component states. The conditions carried by messages that support each component state are selectively and independently checked based on the property to be verified. In other words, we encode clues (conditions) into each component through decoupling. Then, through intra-component independent reasoning, clues that support the control plane state of the component are integrated. Then, in the verification, the clues supporting each property are integrated and are independently checked to complete the ensemble verification. The route redistribution can be simply considered by introducing additional edges between components (§4).

The decoupling principle in the control plane reduces the impact of complex inter-instance and inter-protocol dependencies, enabling scalable and independent control plane state computation. Furthermore, by making the dependencies between routing behaviors explicit, the decouple-then-integrate approach facilitates the construction of causal chains across routing behaviors and the identification of root causes for property violations, significantly improving interpretability.

Key insight 2: independence between the forwarding behaviors of encapsulated and pre-encapsulated packets. In the data plane, packet encapsulation prepends a new header

over the existing one, making the forwarding of the encapsulated packet independent of the pre-encapsulated (decapsulated) packet. Therefore, the procedure where packets are encapsulated into a tunnel within a virtual network N_1, forwarded to the tunnel endpoint through another virtual or physical network N_2, and then decapsulated to reach the next hop in N_1, can be decoupled into two parts: (i) forwarding of the packet with the pre-encapsulated (decapsulated) header in N_1; and (ii) forwarding of the packet with the encapsulated header in N_2. The corresponding forwarding behaviors can then be reasoned independently in N_1 and N_2, with the forwarding behavior in N_2 treated as a dependent condition for the packet in N_1 to successfully reach the next hop. Moreover, this decoupling naturally generalizes to nested encapsulation across multiple virtual networks. For example, if N_2 forwards the packet by further encapsulating it into N_3, the same decoupling procedure applies, and the forwarding in N_3 becomes a dependent condition for that in N_2.

Decoupling in the data plane isolates the complexity introduced by encapsulation and avoids redundant forwarding when multiple virtual networks share the same tunnel. It also reduces the impact of complex dependencies across virtual networks, enabling independent analysis within each virtual or physical network. This not only improves scalability but also makes dependencies between forwarding behaviors explicit, thereby enhancing the interpretability of property violations.

3.2 Workflow of MEV

In this paper, we define each source-destination pair as a tuple $(r_s, vrf_s, r_d, vrf_d, prefix_d)$, where r_s and r_d are the source and destination routers, vrf_s and vrf_d are the corresponding VRFs, and $prefix_d$ is the destination prefix. Since a dependent condition captures the expected forwarding behavior (e.g., reachability) for a specific source-destination pair, it is also defined as $(r_s, vrf_s, r_d, vrf_d, prefix_d)$ (details in §4.1). We use *DEP* to denote the set of all dependent conditions in the network.

An example network. Figure 1a illustrates a typical cloud service provider (CSP) scenario, comprising a provider network ($N1$) and four customer networks ($DC1$, $DC2$, $N2$, and $N3$). In $N1$, BGP EVPN L3VPN over SRv6 is employed to construct virtual networks for different tenants, including $DC1$ and $DC2$ ($N2$ and $N3$). The underlay network runs BGP, ISIS, and SRv6 BE over ISIS. The BGP EVPN peering between PE_1 and PE_2 is established over loopback interfaces, with reachability ensured by ISIS. Between data centers $DC1$ and $DC2$, BGP EVPN L3VPN over VXLAN establishes virtual networks for tenant VMs ($VM1$, $VM3$, and $VM2$, $VM4$). The BGP EVPN peering between CE_3 and CE_4 is also configured over loopbacks, relying on reachability provided by virtual networks constructed by $N1$. BGP peerings between customer and provider networks are established through directly connected interfaces, while all other BGP peerings are configured over loopback interfaces.

Brief overview about dependent conditions. (i) *Control plane.* The BGP EVPN processes on CE_3 and CE_4 depend on the BGP processes on CE_3, PE_1, PE_2, and CE_4, as well as the BGP EVPN processes on PE_1 and PE_2. And the BGP EVPN process on PE_1 and PE_2 further depends on the ISIS process in $N1$. (ii) *Data plane.* The virtual network constructed by the VXLAN tunnel depends on the virtual network built via the SRv6 tunnel in $N1$. Both of them rely on the underlay network.

Property and misconfiguration. To ensure end-to-end connectivity, operators expect reachability from $VM3$ to $VM1$. However, an outbound ACL misconfiguration on PE_1's $port_1$ blocks traffic destined for prefix 2002::/16, causing packet drops inside the SRv6 tunnel. This error spans across routing instances and ultimately leads to unreachability between $VM3$ and $VM1$.

Workflow overview. MEV first decomposes the control plane into disjoint components, one per routing protocol or instance. Routing message propagation is then simulated independently in each component, while recording the communication support required from other components for message validity. Second, MEV generates the forwarding table for each VRF on every router based on the converged control plane states of components. Since the forwarding rules are generated from routing messages, they require the same communication support as the messages used to

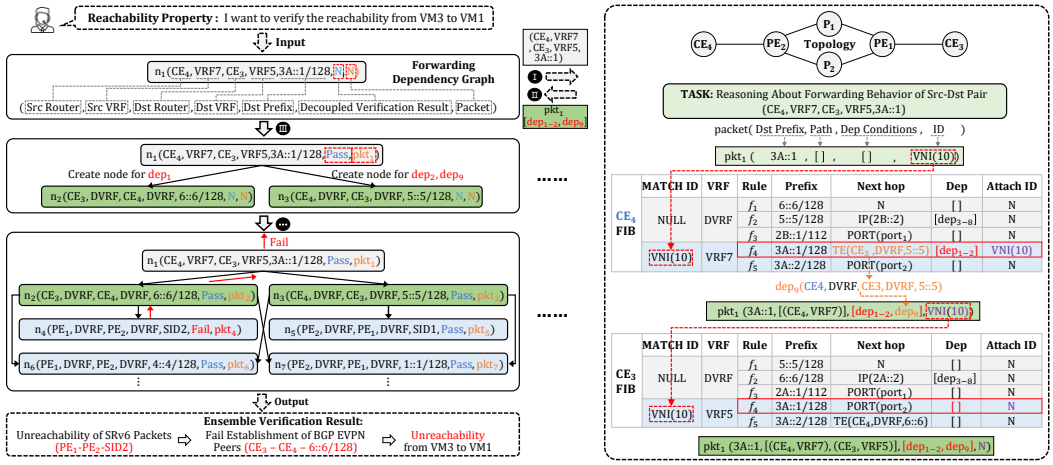
generate them. Third, during packet forwarding, MEV accumulates the supports associated with the matched forwarding rules as the necessary conditions for successful packet delivery. To simplify forwarding across virtual networks, MEV treats forwarding through a tunnel as a shortcut: the packet is directly forwarded to the tunnel endpoint, and the successful delivery of the encapsulated packet is treated as an additional dependent support for the original packet's forwarding. Finally, based on the communication supports required by forwarding behaviors within each virtual or physical network, MEV constructs a forwarding dependency graph (FDG) that explicitly represents the dependent relationships of behaviors across virtual and physical networks. The verification result for each property is then obtained by checking whether all supports in the FDG are satisfied.

(1) Ensemble control plane model construction. To enable independent computation of the control plane states of different routing instances or protocols, MEV constructs an ensemble control plane model for the entire network, consisting of a series of disjoint process components. In each component, vertices represent routing processes, and edges model the exchange of routing messages between them. Unlike existing models, each component not only captures the control plane behaviors of a specific routing instance or protocol, but also encodes the dependent conditions required for communications between processes on corresponding edges. Figure 1c shows the ensemble model of the network in Figure 1a, consisting of several components (i.e., C_1 to C_{16}) across multiple protocol layers. In C_1, the edge between vertices $(CE_3, DVRF, EVPN)$ and $(CE_4, DVRF, EVPN)$ models interactions between the EVPN processes under the default VRF of routers CE_3 and CE_4. The dependent conditions dep_1 and dep_2 represent the communication supports (i.e., reachability between peer IPs) required by the end processes, which are ensured by components C_2, C_6, and C_9. Besides, cross-protocol interactions such as route redistribution are captured via redistribution edges across components. For example, the edge between C_4 and C_1 indicates that messages from the BGP process under $VRF6$ on CE_3 are converted into EVPN messages and then propagated in C_1.

(2) Control plane state computation. MEV independently computes the control plane state of each routing instance or protocol by independently propagating routing messages within each process component. During this procedure, routing messages capture all dependent conditions on its propagation path. And interactions between different instances or protocols can be considered by redistributing messages through edges between components. At convergence, each process vertex maintains a set of routing messages, each annotated with a set of conditions required for ensuring its selection. As illustrated in Figure 3, despite the dependent relationships between C_1 and C_2 as described above, the control plane states of them can be computed independently through propagating messages m_1 and m_2, which carry all dependent conditions (e.g., dep_1 to dep_4) along the propagation path. The message selection of each vertex is governed by the merge function $\oplus$, and routing policy operations are captured by the transfer function trans. (See §4 for details.)

(3) Dependency forwarding table computation. Based on the converged control plane state, MEV computes the dependency forwarding table FIB for each VRF vrf on every router r. Specifically, FIB is constructed by merging all routing messages maintained by each process vertex belonging to the same VRF vrf and router r, together with all connected and static routes configured under the VRF. Each forwarding rule $f \in FIB$ contains a set of dependent conditions D, which is derived from the corresponding routing message, indicating the dependent conditions in the control plane required to generate f. As shown in Figure 2b, the FIB of the default VRF on CE_4 is derived from the connected routes and messages maintained by vertex $(CE_4, DVRF, BGP)$.

(4) Ensemble reasoning for forwarding behavior. Based on the physical network topology and dependency forwarding tables, MEV performs ensemble reasoning for the forwarding behaviors of the source-destination pair $src\text{-}dst_p$ specified by the property to be verified. Throughout the procedure, MEV employs the forwarding dependency graph (FDG) to capture the dependent relationships between forwarding behaviors of different source-destination pairs, which are derived



(a) Forwarding dependency graph (FDG) construction and ensemble verification result computation. Each FDG node captures the forwarding behavior of a source-destination pair within a virtual or physical network. The black edge ⟶ in FDG denotes a dependent relationship between forwarding behaviors. The red edge ⟶ indicates the causal chain of the violation. ($SID1$ = 2001::1, $SID2$ = 2002::1.)

(b) Reasoning for the forwarding behavior of a source–destination pair within a single virtual network. ⇢ denotes the matching process between the packet and the VRF according to their VNIs. □ highlights the rule matched by the packet. ⇢ indicates the generation of dependent conditions derived from tunneling.

Fig. 2. Ensemble verification procedure for a reachability property.

from properties or dependent conditions. Each FDG node contains two main fields: (i) *src-dst*, which denotes the source-destination pair; and (ii) *pkt*, which captures the packets received by the correct or incorrect destinations during forwarding. Besides, FDG maintains a node list, *unReason*, including all nodes whose forwarding behaviors for the source-destination pairs need to be reasoned.

To perform ensemble reasoning for the forwarding behavior of $src\text{-}dst_p$, MEV first creates a node n_1 in FDG for $src\text{-}dst_p$. If FDG already contains equal nodes, MEV finishes the reasoning. Otherwise, MEV adds n_1 to the *unReason*. Then, MEV performs packet forwarding for each source-destination pair $src\text{-}dst_u$ in the FDG node n_u within *unReason*.

Each packet *pkt* additionally maintains its forwarding path and a set of dependent conditions D required for its successful forwarding. As the packet is forwarded, the dependent conditions of any matched forwarding rule f are added to the dependent conditions D of the packet. Besides, to decouple forwarding across virtual or physical networks, if the next hop of f is a tunnel in virtual network N_1, MEV forwards the *pkt* directly to the router where the opposite tunnel endpoint is located and then continues the forwarding within N_1. Meanwhile, the forwarding of the encapsulated packet through another network N_2 between the two tunnel endpoints is treated as a dependent condition $(r_s, vrf_s, r_d, vrf_d, prefix_d)$ for the forwarding in N_1. This new condition is added to the conditions D carried by *pkt*. The r_s (r_d) and vrf_s (vrf_d) represent the router and VRF where the source (destination) endpoint is located, and $prefix_d$ is the encapsulated destination IP.

Finally, MEV collects the packet pkt_r received by the destination specified by $src\text{-}dst_u$ (or the incorrectly reached destination) and updates n_u with it. MEV then creates nodes for source-destination pairs specified by dependent conditions in $pkt_r.D$, treating them as children of n_u. Any nodes not contained in the FDG are added to *unReason*. MEV repeats this procedure until *unReason* is empty.

As shown in Figure 2a, the FDG is constructed during ensemble reasoning for the forwarding behavior of the pair $(CE_4, VRF7, CE_3, VRF5, 3A::1/128)$. The FDG captures all forwarding behaviors required for reachability between the source and destination, along with their dependent relationships. In particular, node n_3 is derived from the dependent conditions dep_2 and dep_9 carried by

packet pkt_1. Here, dep_2 is from the control plane as described above. And dep_9 captures a data plane dependency across virtual networks, ensuring the reachability of VXLAN-encapsulated packets.

(5) Ensemble verification result computation. Based on the verifying property, MEV first checks the pkt field in each FDG node to compute the decoupled verification result (Pass/Fail) for them. Then, leveraging the relationships between FDG nodes, MEV computes the ensemble verification result for each node by integrating the decoupled verification results of its child nodes. For example, as shown in Figure 2a, when verifying reachability from $VM3$ to $VM1$, the decoupled result for node n_4 is Fail (False), as SRv6 packet pkt_4 is dropped due to an ACL misconfiguration on PE_1's $port_1$. By integrating the decoupled verification results, MEV produces a final ensemble result (Fail) of node n_1, which corresponds to the reachability property under verification. MEV identifies the root cause and constructs a causal chain for the violation by searching the FDG. (Details in §5.3)

Moreover, benefiting from the decoupling in the control and data planes, MEV confines recomputation to only the affected routing instances or virtual networks when configuration updates occur. It avoids re-reasoning about the behavior of elements that depend on–or are depended upon by–the affected elements, enabling fast re-verification when configuration updates. (Details in §5.4.)

4 Control plane reasoning

In this section, we introduce the ensemble control plane model, which is composed of a series of process components, along with its semantics.

4.1 Dependency process component

Each dependency process component corresponds to a routing instance or protocol. Formally, a component is defined as a tuple $C = (V, E, DEP, M, \oplus, \text{dep}, \text{trans}, \text{init})$, where:

- V is a set of vertices, each representing a routing process of a specific routing instance or protocol configured under a particular VRF on a router. Each vertex $v \in V$ is defined as a tuple $(r_l, vrf_l, type)$, where r_l and vrf_l denote the router and VRF where the process locates, and $type$ specifies the type of routing process (e.g., BGP, EVPN).
- $E : V \times V$ represents a set of directed edges between the vertices, capturing the interactions (e.g., routing message exchanging) between corresponding routing processes.
- DEP is a set of dependent conditions as defined in §3.2. And parts of DEP are required for interactions between routing processes within the routing instance or protocol.
- M is the set of routing messages exchanged between routing processes. Each message $m \in M$ is defined as a tuple $(prefix, ad, attr, nh, D)$. The $prefix$ specifies the destination prefix, and ad represents the administrative distance. The $attr$ is a tuple with multiple fields, representing the protocol-specific attributes. Its structure varies in different components. For example, in C_1 shown in Figure 3, $attr$ in the message is defined as $(extcomm, asp, id)$, where $extcomm$ includes extended communities (e.g., route targets), asp represents the AS path, and id denotes network identifiers (e.g., VNI, SID) associated with each VRF. And nh indicates the next hop, which can be: (i) a concrete IP address, which is used for underlay routing messages. (ii) a tunnel endpoint represented as $TE(r_t, vrf_t, IP_{en})$, which is specific to overlay networks and indicates that packets matching m should be encapsulated into the tunnel. The r_t and vrf_t represent the router and VRF where the tunnel endpoint resides, and IP_{en} is the outer header used for encapsulation. (iii) an output port on a router. Finally, $D \in 2^{DEP}$ is the set of dependent conditions accumulated during the message propagation, specifying requirements for the message's validity.
- $\oplus : M \times M \rightarrow M$ is the merge function that combines two different routing messages (e.g., from different peers) that share the same $prefix$ (and rd in the $attr$), selecting the better one based on the $attr$ carried by them. Here, rd refers to the route distinguisher.
- $\text{dep} : E \rightarrow 2^{DEP}$ maps each edge $e(u, v)$ to a set of dependent conditions required for the successful exchange of routing messages between the endpoint process vertices. Routing processes establish

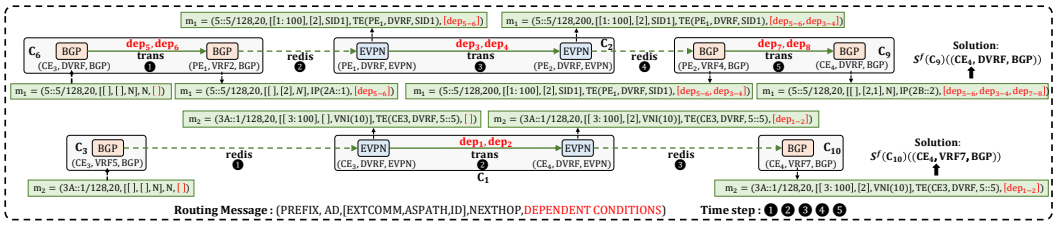


Fig. 3. Partial procedure of computing the control plane state within the ensemble control plane model. The edge $\longrightarrow$ denotes the exchange of messages between processes within the same protocol or instance, while the edge $\dashrightarrow$ indicates message redistribution across different protocols or instances. trans and redis denote the transfer and redistribution functions applied to messages.

peer sessions using mechanisms like HELLO messages (e.g., ISIS, OSPF) or TCP connections (e.g., BGP, BGP EVPN), enabling the exchange of routing messages. This exchange depends on the reachability of the HELLO or TCP messages, supported by other routing instances or protocols. Thus, for edge $e(u, v) \in E$, $\mathsf{dep}(e)$ contains two dependent conditions, one for each reachability direction. For a condition $dep(r_s, vrf_s, r_d, vrf_d, prefix_d) \in \mathsf{dep}(e(u, v))$, r_s and r_d are the locating routers r_l of u and v, respectively, while vrf_s and vrf_d are the locating VRFs vrf_l. The $prefix_d$ is determined as follows: (i) For implicitly configured and automatically discovered peers (e.g., OSPF and ISIS peers), $prefix_d$ is a 128-bit IPv6 prefix derived from the IP of the interface that connects r_s and r_d and runs the process. (ii) For explicitly configured peers (e.g., BGP and BGP EVPN peers), $prefix_d$ is derived from the peer IP configured on the process corresponding to u for reaching v. The dependent condition for the direction from v to u is set similarly.

- $\mathsf{trans} : E \times M \longrightarrow M$ is the function that transforms the routing message m as it traverses the edge $e(u, v)$. There are mainly three types of operations in each trans: (i) updating the next hop nh of m according to process vertex u. (ii) modifying the attribute $attr$ or block m according to the export routing policies of u and import policies of v (as shown in Figure 3), (iii) updating the dependent conditions D carried by m by adding the dependent conditions $\mathsf{dep}(e)$. We use the notation trans_{uv} to refer to the transfer function between u and v.
- $\mathsf{init} : V \longrightarrow M$ is the initialization function that assigns each process vertex $v \in V$ with the routing messages initially originating from it.

4.2 Ensemble control plane model

Then, we use all dependency process components constructed from the network to build the ensemble control plane model, which is defined as a tuple $N = (\bigcup_{i=1}^{n} C_i, RE, \mathsf{redis})$, where:

- $\bigcup_{i=1}^{n} C_i$ represents the set of all dependency process components derived from the configuration of the whole network. Each C_i is a tuple $(V_i, E_i, DEP_i, M_i, \oplus_i, \mathsf{dep}_i, \mathsf{trans}_i, \mathsf{init}_i)$.
- $RE : V_i \times V_j$, where $V_i \in C_i$, $V_j \in C_j$, and $i \neq j$, is a set of redistribution edges. Each edge $re \in RE$ captures the interactions (e.g., route redistribution) between routing processes in different protocols. For each $re(u, v) \in RE$, it is required that the $types$ of u and v must be distinct.
- $\mathsf{redis} : RE \times M_i \longrightarrow M_j$ is the function that transforms routing messages across protocols according to redistribution policies. For each edge $re(u, v) \in RE$ with $u \in V_i, v \in V_j$, the function $\mathsf{redis}_{uv}(m_i)$ takes $m_i \in M_i$ and produces $m_j \in M_j$ such that: (i) $m_j.prefix$ equals $m_i.prefix$; (ii) $m_j.ad$ and $m_j.attr$ are initialized to the default administrative distance and attributes of the protocol indicated by $v.type$; (iii) If v corresponds to an overlay process and u represents an underlay process, $m_j.nh$ is updated to a tunnel endpoint, indicating the tunnel constructed by the message. For example, in Figure 3, the nh of message m_1 is updated to the tunnel endpoint $TE(PE_1, DVRF, SID1)$ when it is propagated from $(PE_1, VRF2, BGP)$ to $(PE_1, DVRF, EVPN)$. Otherwise, $m_j.nh$ is set to Null. (iv) Redistribution policies between processes u and v match m_i, possibly modifying or blocking the message m_j.

For example, Figure 1c shows the ensemble control plane model of the network in Figure 1a.

Given the ensemble model N, its semantics are defined as $\sigma = (N, \mathcal{T}, \mathcal{S}, \tau, \text{choice})$, where:

- $\mathcal{T}$ is a set of time steps.
- $\mathcal{S} : C_i \to (V_i \to M_i)$ defines the control plane state of component C_i in N, where V_i is the set of vertices and M_i is the set of routing messages associated with C_i. More specifically, $\mathcal{S}^t(C_i)(v)$, where $t \in \mathcal{T}, v \in V_i$, denotes the set of routing messages selected by v by time t.
- $\tau : \mathcal{T} \to 2^{\bigcup_{i=1}^{n} V_i}$ is a time schedule that maps a time step to the set of vertices that process routing messages at this time step.
- $\text{choice} : \mathcal{T} \to V_i \to V_j \to M_j$ is a message flow function where $\text{choice}^t(u, v)$ gives routing messages sent by $u \in V_i$ to $v \in V_j$ at time t. Specifically, the $\text{choice}^t(u, v)$ is defined as follows:

$$\text{choice}^t(u,v) = \begin{cases} \text{trans}_{uv}(\mathcal{S}^{t-1}(C_i)(u)), & \text{if } i = j, \\ \text{redis}_{uv}(\mathcal{S}^{t-1}(C_i)(u)), & \text{if } i \neq j, \end{cases}$$

The control plane state of each vertex $v \in V_i$ at time t, where V_i belongs to C_i, is computed as:

$$\mathcal{S}^0(C_i)(v) = \text{init}_i(v)$$

$$\mathcal{S}^t(C_i)(v) = \begin{cases} \mathcal{S}^{t-1}(C_i)(v), & \text{if } v \notin \tau(t), t > 0 \\ \mathcal{S}^0(C_i)(v) \oplus \left(\bigoplus_{(e(u,v) \in \cup_{i=1}^{n} E_i) \cup (re(u,v) \in RE)} \text{choice}^t(u,v) \right), & \text{if } v \in \tau(t), t > 0. \end{cases}$$

At the initial step, $t = 0$, each vertex v in N is initialized with the routing messages originating from it. For $t > 0$, if vertex v is scheduled to update, it merges all routing messages received from its neighboring vertex u, denoted as $\text{choice}^t(u, v)$. These messages are derived from u's control plane state at the previous step, $t - 1$. If u and v belong to the same component C_i, all routing messages in $\mathcal{S}^{t-1}(C_i)(u)$ are processed by the transfer function trans_{uv} corresponding to the edge $e(u, v) \in E_i$. Otherwise, they are processed by the redistribution function redis_{uv} for the redistribution edge $re(u, v) \in RE$. If v is not scheduled for an update, its state remains unchanged. The process continues until the control plane reaches a stable state, $\mathcal{S}^f$, where the state of each vertex v in N at time step t equals its state at $t - 1$. For example, Figure 3 shows the procedure of computing routing messages for different components. Although component C_1 depends on C_2, C_6, and C_9, the control plane states of them can be computed independently.

5 Property ensemble verification

We then explain how to use the converged control plane state, $\mathcal{S}^f$, to check whether the actual forwarding behavior of a specific source-destination pair satisfies the property to be verified.

5.1 Dependency forwarding table computation

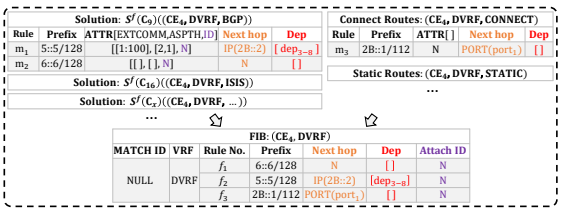


Fig. 4. Dependency forwarding table computation.

Each router maintains multiple VRFs, each associated with its own forwarding table and a set of network identifiers (e.g., VNI, SID) denoting virtual networks constructed by it. Upon receiving a packet, the router selects a VRF based on the packet's network identifier and performs forwarding according to the VRF's forwarding table. Therefore, MEV computes a dependency forwarding table for each VRF vrf on router r, leveraging the control plane state $\mathcal{S}^f$ and the connected (and static) routes configured on the router. Besides, MEV maintains an ACL list for each port of every router.

Each rule f in the forwarding table is defined as a tuple $(match, nh, D, id)$. The $match$ specifies the packet matching criteria. For simplicity, it is referred to as $prefix$ in the following. And other criteria (e.g., MPLS label) can be considered through extension. The nh denotes the next hop, and $D \in 2^{DEP}$ is a set of dependent conditions required to generate f. The definitions of nh and D are consistent with those in §4.1. The id indicates the network identifier applied to packets matching f, determining the virtual network (i.e., VRF) to which the packets will be forwarded at the next hop router.

The forwarding table FIB of VRF vrf on router r is computed in three steps (Algorithm 1 in Appendix D): (i) Select the set V_F of vertices in the ensemble control plane model, whose locating router r_l and VRF vrf_l match r and vrf. (ii) Merge all routing messages in the $\mathcal{S}^f(C_i)(v)$, where $v \in V_F$ and C_i is the component containing v, generating the routing table RIB for the vrf. Besides, the connected and static routes configured under the vrf are also merged. (iii) Convert each $m \in RIB$ to the forwarding rule $f \in FIB$. For each f, the $match$, nh, and D are directly derived from m. The id is derived from the $attr$ of m. If no network identifier is present in $attr$, the id is set to Null. For example, Figure 4 shows the procedure of computing the FIB for the default VRF ($DVRF$) on CE_4.

5.2 Ensemble reasoning for forwarding behavior

Then, we first introduce the structure of the forwarding dependency graph (FDG), which captures dependent relationships among forwarding behaviors of different source-destination pairs (from properties and dependent conditions). We then illustrate how to perform ensemble reasoning for the source-destination pair $src\text{-}dst_p$ specified by the property to be verified with updating the FDG.

5.2.1 Forwarding dependency graph

The forwarding dependency graph (FDG) consists of three parts: (i) **FDG node list**. Each FDG node n is represented as a tuple $(src\text{-}dst, child, parent, deResult, pkt)$, where $src\text{-}dst(r_s, vrf_s, r_d, vrf_d, prefix_d)$ specifies the source-destination pair. The $child$ tracks FDG nodes depended on by n, meaning the establishment of forwarding behavior of $src\text{-}dst$ depends on the behavior of the source-destination pairs specified by nodes in $child$. Similarly, $parent$ tracks the nodes that depend on n. The $deResult$ represents the decoupled verification result (Pass/Fail), indicating whether the forwarding behavior of $src\text{-}dst$ in a specific virtual or physical network satisfies the property. Finally, pkt is the packet received by the destination specified by $src\text{-}dst$ or the destination to which it is incorrectly forwarded. (ii) **Hash-radix trie dual indexing**. The first-level index maps a hash value derived from the source-destination string, which includes r_s, vrf_s, r_d, and vrf_d, to a radix trie. The second-level index employs the radix trie, where each edge contains binary bits from the $prefix_d$. Each trie node stores a bit string that consists of binary bits on the path from the root to it, representing the matched prefix $prefix_d$. Besides, each trie node stores the index of the FDG node in the node list, whose source-destination pair matches the $src\text{-}dst$ identified by the dual indexing. (iii) **Unreasoned node list**. The $unReason$ maintains all FDG nodes, whose forwarding behaviors need to be reasoned.

When adding a new FDG node n, the FDG first traverses the dual indexing based on the $src\text{-}dst$ of n to check: (i) whether there exists a node n_c that covers n (i.e., n_c has the same source and destination routers and VRFs as n, and a parent destination prefix of that in n); (ii) whether there exists a node n_e that is exactly equal to n (i.e., n_e has the same source-destination pair as n). The FDG is then updated as follows: (i) If n_e exists, n is discarded. (ii) If n_c exists, n is added to the FDG node list but not to $unReason$. (iii) Otherwise, n is added to both the FDG node list and $unReason$. In all cases where n is added to the FDG node list, the dual indexing is updated accordingly.

Using dual indexing, the FDG identifies the existence of n_e and n_c for node n in $O(k)$ time, where k is the length of the destination prefix in n, avoiding redundant forwarding behavior reasoning. Note that MEV constructs an FDG for all properties.

5.2.2 Ensemble reasoning with updating FDG

Each packet is defined as a tuple $(prefix, P, D, id)$. The $prefix$ is the destination prefix, and P indicates the forwarding path consisting of a list of tuples (r, vrf) where each tuple represents the router and VRF traversed by the packet. $D \in 2^{DEP}$ is the set of dependent conditions required for the packet to be forwarded along P. And id is the network identifier assigned to the packet.

Ensemble reasoning for pair $src\text{-}dst_p$ begins by resetting the FDG's $unReason$ list, creating a new node n for the pair, and adding it to the FDG. MEV then independently processes each node $n_u \in unReason$ (Algorithm 1 in Appendix D) as follows until the list is empty.

(1) Reasoning about the forwarding behavior of the pair $src\text{-}dst_u(r_s, vrf_s, r_d, vrf_d, prefix_d)$ in n_u. MEV first generates an initial packet pkt, where $prefix$ is equal to $prefix_d$, and both P and D are set to empty. The id of pkt is set to the network identifier associated with source VRF vrf_s, or Null if vrf_s is the default VRF ($DVRF$). For example, in Figure 2b, for the pair $(CE_4, VRF7, CE_3, VRF5, 3A{::}1)$, pkt_1 is generated with id initialized to $VNI(10)$, corresponding to the VNI associated with $VRF7$ on CE_4. The packet is then injected into source router r_s for forwarding.

During forwarding, each router r performs the following steps: (a) Direct packet pkt to the VRF vrf whose network identifier matches packet's id. If the packet's id is Null, the default VRF is used. And the tuple (r, vrf) is appended to the forwarding path P of pkt. (b) Forward the packet according to the forwarding table FIB of vrf. For each rule $f \in FIB$ matched by pkt, its dependent conditions are appended to the set of dependent conditions D carried by pkt. Then, pkt is forwarded to the next hop nh specified by f. If nh is a port, pkt is processed (i.e., permitted/blocked) by the ACL list associated with that port. If nh is a tunnel endpoint $TE(r_t, vrf_t, IP_{en})$, pkt is directly forwarded to the router r_t hosting the tunnel endpoint. And a new dependent condition $dep(r_s, vrf_s, r_d, vrf_d, prefix_d)$ is generated and added to the set of dependent conditions D in pkt, where the source router r_s is set to the router r where pkt currently locates. The destination router r_d is set to the r_t where the tunnel endpoint is located. Both the source VRF vrf_s and destination VRF vrf_d are set to the default VRF $DVRF$, since the encapsulated packet is typically forwarded based on the default VRF. The $prefix_d$ is set to the encapsulated IP IP_{en}, which is the outer IP address of the encapsulated packet.

(2) Updating the FDG. First, MEV adds the received packet pkt_r to the packet field of node n_u, where pkt_r is the packet received by the destination VRF on the destination router (or the incorrectly reached VRF on a specific router). Next, MEV creates a new FDG node n_d for the source-destination pair specified by each dependent condition $dep \in pkt_r.D$, and adds it to the FDG. Furthermore, n_d is considered as a child node of n_u. If there is an equal node n_e in FDG, n_d is replaced with n_e.

For example, Figure 2 illustrates the whole procedure about ensemble reasoning for the forwarding behavior of the source-destination pair $(CE_4, VRF7, CE_3, VRF5, 3A{::}1/128)$.

5.3 Ensemble verification result computation

As shown in Table 6, MEV supports verifying many routing and forwarding properties. Without loss of generality, we take the most basic forwarding property–reachability–as an example. (Others are discussed in Appendix B.) First, MEV applies the check function to the packet field pkt of each node n, computing the decoupled verification result $deResult$ for n. The check function validates whether the forwarding behavior of the packet conforms to the target property. For reachability, check returns True if the last router-VRF pair in the forwarding path of pkt matches the destination (r_d, vrf_d) specified by $n.src\text{-}dst$. Then, MEV computes the ensemble verification result for each node n as: $\mathsf{enResult}(n) = n.deResult \wedge (\wedge_{n_i \in n.child} \mathsf{enResult}(n_i))$. For each violated property, MEV runs a depth-first search (DFS) from the corresponding FDG node to collect all descendant nodes whose ensemble result is False, forming a violation subtree. Leaves in the subtree indicate the root causes of the violation, and paths from leaves to the root capture causal chains explaining how the

violation arises. Since the computational complexity of DFS is manageable, MEV can easily find root causes and generate interpretable causal explanations, even in large and complex networks.

For example, as shown in Figure 2a, the Fail (False) of n_4 (due to the error ACL) results in the unreachability from $VM3$ to $VM1$. Both the root cause and the causal chain are provided.

Discussion. MEV achieves high interpretability by embracing the principle of ensemble verification and explicitly constructing the FDG. This design introduces a natural trade-off between the complexity of the FDG and the level of interpretability provided by MEV. However, we observe that this trade-off remains well-balanced in practice: the complexity of the FDG always stays within acceptable bounds. For a network, suppose there are M_p properties to be verified. Let M_s denote the number of peer relationships established among routing processes, and let M_t denote the number of tunnels constructed after network convergence. Then, the maximum number of nodes in the FDG is bounded by $O(M_p + M_s + M_t)$. Moreover, the average time complexity for querying an FDG node, as well as for adding FDG nodes or specific edges in FDG, is $O(k)$, where k denotes the prefix length of the destination prefix $prefix_d$ in the *src-dst* associated with the given node or edge.

Correctness of MEV. We discuss the correctness of MEV in Appendix A.

5.4 Verification under configuration update

We next describe how MEV efficiently updates verification results under configuration updates with minimal recomputation.

(1) Limited re-computation of control plane states. MEV only recomputes the behavior of the process components whose configurations update and the components interacting with them, which involves: (a) Identify the vertex v whose configuration updates and its component C_i. (b) Reinitialize the control plane state of v to $\mathsf{init}_i(v)$, keeping all other vertices u at their previously converged states $\mathcal{S}^f(C_j)(u)$. (c) Incrementally recompute the control plane state of the ensemble model N according to its semantics. (d) Ascertain the set V_{up} of vertices whose states have changed.

(2) Limited re-reasoning of forwarding behavior. MEV limits the re-reasoning of forwarding behavior to the affected virtual or physical networks as follows: First, for each updated vertex $v \in V_{up}$ that locates on router r_l and VRF vrf_l, MEV updates the forwarding table of vrf_l and adds (r_l, vrf_l) to the update list $RVRF_{up}$. Second, MEV identifies the set FDG_{up} of FDG nodes whose forwarding behaviors need to be re-reasoned. For each $(r, vrf) \in RVRF_{up}$, MEV collects a set $FDGN_t$ of FDG nodes, each of which contains packets traversing (r, vrf). It then checks whether any changed rule in the FIB of vrf overlaps with the destination prefix in the source-destination pair of a node $n \in FDGN_t$. If so, n is added to $FDGN_{up}$. Finally, MEV performs ensemble reasoning for each source-destination pair specified by the node in $FDGN_{up}$ and updates the FDG.

(3) Update verification result. MEV incrementally updates the verification results for each property by performing the procedure as described in §5.3.

6 Evaluation

We are primarily interested in evaluating MEV in the following dimensions: (i) the efficiency of MEV in the NOC of the real nationwide network, FITI [22]; (ii) the performance of MEV compared with extensions of state-of-the-art verifiers; (iii) the effectiveness of the FDG in MEV. All experiments are run on a Linux server with two 24-core AMD EPYC 7402 CPUs@3.3GHz and 256GB memory.

6.1 Benchmark networks

6.1.1 Real nationwide network - FITI

The Future Internet Technology Infrastructure (FITI) [22] is a nationwide IPv6-based network spanning 40 universities across 35 cities in China, with 80 nodes. FITI creates 4096 virtual networks on top of the physical IPv6 network, with each virtual network connecting 40 customer networks. FITI utilizes two distinct methods to construct these virtual networks: (i) white-box method, which

Category	Size	Routers	Leafs (VPCs)	Subnets (VRFs)
Three-Tier Clos DCNs	Small	5	2	4
		20	8	64
		45	18	324
		80	32	1024
	Medium	125	50	2500
		180	72	5184
		245	98	9404
	Large	320	128	16384
		405	162	26244
Two-Tier Clos DCNs	Small	8	6	120
		12	10	200
		22	20	400
		52	50	1000
	Medium	102	100	2000
		202	200	4000
	Large	502	500	10000
ISPs	Small	49	30	900
	Medium	86	50	2500
	Large	174	100	10000

Table 1. DCNs and ISP networks of various scales

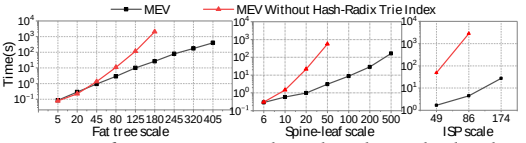


Fig. 5. Verification time with and without dual index

Network	Scale	Routers	No. Conditions to be Verified		Reduce Percent
			MEV w/o FDG	MEV	
ISP	Small	49	52,200	2,352	95.5%
	Medium	86	245,000	7,310	97.0%
	Large	174	1,980,000	30,102	98.5%
Two-tier DCNs	Small	8	3,600	110	96.9%
	Medium	102	396,000	10,100	97.4%
	Large	502	9,980,000	250,500	97.5%

Table 2. Number of conditions to be verified

Update	Explanation
Add VPC	Add a VPC with 10 customers to random leaves
Add Subnet	Add a customer and enable specific VPC access
Modify Access	Change VPCs' access to a subnet

Table 3. Types of synthesized updates

creates 2048 virtual networks using VRF in the ISIS protocol; and (ii) black-box method, which builds another 2048 virtual networks using BGP EVPN L3VPN over SRv6. In this setup, ISIS ensures communication for the underlay, while SRv6 tunnels ensure communication within the overlay.

6.1.2 Complex data center, ISP and hybrid networks

To evaluate MEV's performance in real-world scenarios, we analyzed industry-standard DCN and ISP configurations (e.g., Google, Tencent) and benchmark networks from previous works [42]. We synthesized networks with small, medium, and large topologies and virtual network sizes, as shown in Table 1. DCNs followed the widely used Clos topology [43], while ISPs utilized topologies from the Topology Zoo dataset [44]. Each network emulates multi-tenant environments with VPCs or VPNs to support distributed tenant connectivity and enforce isolation.

(1) Data center networks: We employed two common types of Clos topologies: (a) a three-tier design with core, aggregation, and leaf routers, where the core layer consists of 1 to 100 routers, and both the aggregation and leaf layers range from 2 to 200 routers; (b) a two-tier design with spine and leaf routers, where the spine layer contains 2 routers, and the leaf layer scales from 6 to 500 routers, following the industry standards outlined in PPV [42]. For each DCN with n leaf routers, n virtual networks are created via EVPN L3VPN over VXLAN, each corresponding to a tenant. Each virtual network contains n customer networks, evenly distributed across the leaf routers.

(2) ISP networks: We synthesized ISP networks of varying scales–49, 86, and 174 routers–and configured EVPN L3VPN over VXLAN to create virtual networks for 30, 50, and 100 tenants. Each VPC includes 30, 50, or 100 customer networks, which are randomly distributed in the network. For example, there are 100 VPCs with a total of 10,000 customer networks in the large ISP network.

(3) Hybrid networks: To evaluate the limits of MEV's interpretability, we constructed highly complex networks that integrate both ISP and DC architectures. The networks comprise up to 15 stacked layers of virtual networks and incorporate diverse protocols, including BGP, ISIS, BGP EVPN L3VPN over SRv6, and BGP EVPN L3VPN over VXLAN, exhibiting a level of complexity beyond any other network known to us.

6.2 Different implementations

MEV with new parser. We implemented an MEV prototype comprising ~8000 lines of Java code, supporting both IPv4 and IPv6 networks. MEV handles key control and data plane protocols, including ISIS, BGP, EVPN L3VPN, VXLAN, and SRv6. To meet the requirements for verifying

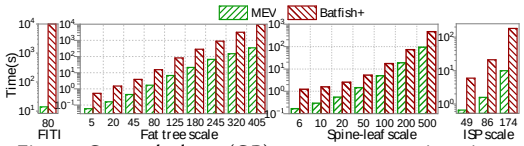


Fig. 6. Control plane (CP) state computation time

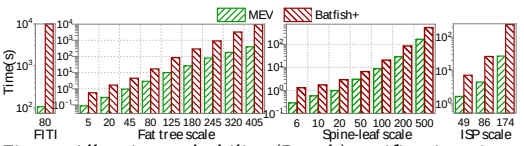


Fig. 7. All-pair reachability (Reach) verification time

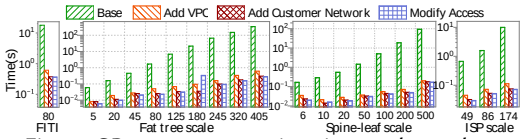


Fig. 8. CP state computation time under update

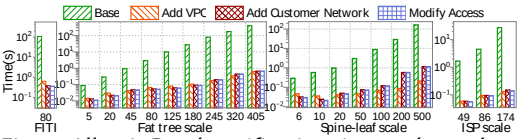


Fig. 9. All-pair Reach verification time under update

FITI, we developed a new configuration parser in MEV for Huawei [45] device syntax. To make verification results more user-friendly in large networks, we enhanced MEV's presentation of property violations. MEV first classifies all violations by their root causes. For each root cause, MEV outputs (i) a concise summary showing the root cause and the set of affected properties, and (ii) an optional causal graph that compactly represents all violations stemming from the same root cause.

Batfish$^+$. To compare MEV with state-of-the-art verifiers, we extended Batfish, one of the most representative monolithic verifiers, to support EVPN L3VPN over VXLAN (SRv6), termed as Batfish$^+$. The original Batfish supports computing control plane states for networks with EVPN L3VPN, but it cannot reason about forwarding behaviors that involve packet encapsulation. We therefore implemented explicit support for encapsulation and decapsulation: packets are encapsulated using a stack-based representation, forwarded through the tunnel, and then decapsulated at the tunnel endpoint, faithfully simulating how encapsulation occurs in real networks. Apart from this extension, Batfish$^+$ reuses the same implementation logic as MEV for forwarding before and after encapsulation, but without constructing the FDG. Importantly, we also applied the same performance optimizations (i.e., parallelization and data structures) to Batfish$^+$ as in MEV.

We did not include other verification tools as baselines for the following reasons. Tools such as HoYan [13] and SRE [16] require additional symbolic manipulations (e.g., conjuncting symbolic link variables with symbolic topology conditions carried by messages) even in failure-free settings. This overhead limits their scalability and makes them theoretically less efficient than Batfish under our evaluation scenarios. For SMT-based tools like Minesweeper [11], the number of constraints grows prohibitively in complex overlay/underlay networks (e.g., exceeding 5 million in our medium ISP topology), rendering them intractable for state-of-the-art solvers (e.g., Z3 [46]). Consistent with this, Figure 13 (Appendix C) shows that when verifying networks without link failures, Tiramisu [12] and Minesweeper [11] perform similarly to, or even worse than, Batfish. Modular verification tools such as Lightyear [21] and Timepiece [20] face inherent challenges in achieving full automation and in verifying forwarding properties, as discussed in §2.2. Therefore, they are also excluded.

6.3 Efficiency of MEV on FITI

Over five months of deployment at FITI, MEV detected 13 reachability violations within virtual networks and 9 isolation violations across them. Below, we highlight three real-world examples of "update errors" identified by MEV during routine operations.

(1) Preventing outage within specific virtual networks. During a routine network upgrade, operators aimed to create a new virtual network for a tenant. Following the FITI configuration standards, they used the extended community "38272:2970" as the export and import route targets (RT) for the new virtual network. However, a legacy routing policy on a provider edge (PE) router mistakenly filtered routes with the community "38272:297[0-9]". While this had no impact before the update, committing the new configurations would prevent customers connected to the new virtual network via this PE router from receiving routes from other customer networks, causing an

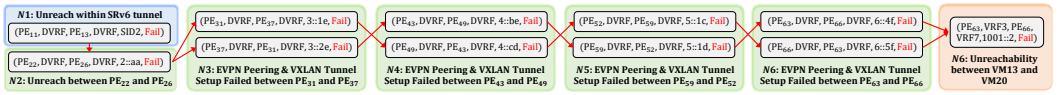


Fig. 10. A real causal chain of errors spanning routing instances across networks ($N1$ to $N6$).

outage. Although operators strive to remove legacy configurations after each update, the complexity of configurations (over 120k lines) makes it difficult to ensure no errors remain. MEV detected this issue in pre-deployment verification, allowing operators to correct it before rollout.

(2) Preventing widespread customer communication outages due to PE-localized misconfiguration. In a recent update, operators attempted to filter all packets in a specific SRv6 tunnel with the SID "2001:253:1:E01::56/128" on a designated port of a PE router for security. However, the prefix length of the ACL was mistakenly set to 96, which would inadvertently affect all SRv6 tunnels under the locator "2001:253:1:E01::/96" on the PE router. If applied, this would disrupt communication between customers in different virtual networks accessing through the PE router and other customers. MEV detected this error by verifying reachability within each virtual network.

(3) Avoiding isolation violations between virtual networks. To accommodate evolving production needs, operators often connect new customer networks to existing virtual networks by creating sub-interfaces on PE routers and binding them to tenant-specific VRFs. However, in one particular update, they mistakenly bound the virtual sub-interface to a wrong VRF, which would lead to the violation of cross-tenant isolation guarantees. MEV successfully identified this error by verifying isolation between virtual networks before applying the update.

6.4 Performance of MEV

We further evaluated MEV's interpretability and effectiveness through a red team–blue team exercise. The red team injected misconfigurations at various layers of the virtual network in the hybrid networks, causing unreachability of specific source-destination pairs in the topmost virtual network. The blue, unaware of the errors, used MEV to diagnose and correct the resulting issues. Figure 10 shows a real causal chain returned by MEV in the complex scenarios. Feedback from the blue shows that: (i) MEV's root cause reports enable precise error localization, allowing operators to quickly pinpoint misconfigurations; and (ii) even when errors span multiple routing instances and virtual networks, MEV's causal chains clearly expose how high-level properties are violated, helping operators validate the correctness of dependent relationships across different virtual networks.

Performance on FITI. Figure 6 compares control plane computation times of MEV and Batfish$^+$ on FITI, and Figure 7 shows end-to-end verification times. Batfish$^+$ fails to complete within the time limit, whereas MEV finishes control plane simulation in under 20s and completes full verification in under 100s. This significant gap demonstrates MEV's advantage in handling large-scale, tenant-dense networks like FITI and highlights a key motivation behind MEV. Moreover, MEV completes incremental verification for all updates in Table 3 in less than 1s, as shown in Figures 8 and 9. The improved scalability makes continuous verification for large and complex networks, such as FITI. This capability not only enables operators to quickly iterate on network configurations during the design phase, but also allows them to swiftly respond to failures during operation.

Performance on synthetic networks. Figures 6 and 7 further show results across synthetic networks of varying scale. MEV outperforms Batfish$^+$ by 10× on average in three-tier DCNs and ISPs. In two-tier DCNs, the performance gap narrows due to the simpler topology. Incremental verification across all synthetic updates also completes in approximately 1s, often requiring nearly less than 1% of the time needed for full verification on large topologies.

Resource consumption. Figures 11 and 12 in Appendix C illustrate memory usage across all experiments. We also quantify the complexity of each benchmark: up to 30K destinations, 5M source-destination pairs, and 16M routes in the converged control plane state. And we conducted experiments to evaluate the correctness of MEV. Detailed statistics are provided in the Appendix C.

6.5 Effectiveness of FDG

We evaluated the effectiveness of FDG and the impact of the hash–radix trie dual indexing design on all-pair reachability verification. Table 2 shows the number of dependent conditions verified during ensemble reasoning across ISP networks and DCNs of varying scales, comparing MEV with and without the FDG. FDG eliminates up to 95% of redundant condition checks. Figure 5 presents the verification time of MEV with and without dual indexing, showing that dual indexing significantly improves performance as network complexity increases.

7 Discussion

Monolithic verification approaches couple different layers (i.e., routing protocols, routing instances, virtual networks, and physical networks) into a unified model. To reason about network behaviors, they must first compute the behaviors of dependent layers and then iteratively compute those of the upper layers, resulting in high complexity and limited scalability. Modular approaches improve scalability by partitioning a topology into multiple sub-topologies and verifying each independently. However, within each sub-topology, they still perform monolithic reasoning across layers, and thus fail to address the complexity introduced by cross-layer interactions in overlay/underlay networks.

In contrast, MEV introduces a novel ensemble verification paradigm. MEV first analyzes the communication supports that each layer's behaviors possibly require from other layers, and encodes this information into the corresponding layer model. This decouples the entire network into a collection of independent per-layer models, enabling MEV to reason about each layer in parallel while horizontally capturing the cross-layer supports actually needed for every intra-layer behavior to hold. During property verification, MEV vertically associates the dependent relationships among behaviors across different layers, constructing for each intra-layer behavior a cross-layer communication support chain that it relies on. This "decouple first, then integrate" paradigm enables parallel intra-layer reasoning to enhance scalability, while also improving interpretability by explicitly identifying the cross-layer dependencies required for each property–capabilities that neither traditional monolithic nor modular verification approaches can provide.

Limitations and future work. Despite the rise of centrally programmed networks (e.g., SDN or SRv6 with traffic engineering), MEV currently targets networks with distributed control, as these remain dominant in practice and continue to be a source of subtle errors [10]. MEV analyzes only a single possible message interleaving and the corresponding converged control and data plane states, rather than exploring all possible interleavings and the transient states during convergence. Another limitation is that MEV assumes a static network without failures and dynamic behaviors (e.g., dynamic tunnel migration), which are left for future work. Currently, we are planning to extend MEV by symbolizing link states to enable verification under different link failures.

8 Conclusion

MEV is the first control plane verifier designed for overlay/underlay networks and ensures high scalability and interpretability through ensemble verification. MEV first decouples the complex dependent conditions within control and data planes, independently computing the control plane states within each process component and reasoning forwarding behaviors within each virtual or physical network by capturing dependent conditions. MEV then integrates and independently verifies all dependent conditions required for network behaviors specified by the properties, completing the verification. MEV has been continuously used in the maintenance of the FITI. Benefiting from MEV, operators can accurately identify and fix misconfigurations in the scenarios with frequent configuration updates, which are common in FITI to support users' evolving experimental needs.

Acknowledgments. We thank our shepherd and all reviewers for their insightful comments. This work was supported by the National Natural Science Foundation of China 62394322 and the National Key R&D Program of China 2024YFB2906800.

References

[1] R Aggarwal, N Bitar, A Isaac, J Uttaro, J Drake, and W Henderickx. Rfc 7432: Bgp mpls-based ethernet vpn, 2015.

[2] J De Clercq, D Ooms, M Carugi, and F Le Faucheur. Rfc 4659: Bgp-mpls ip virtual private network (vpn) extension for ipv6 vpn, 2006.

[3] Ghulam Bahoo and Hafiz Mati Ur Rahman. Segment routing over ipv6 (srv6).

[4] Pankaj Garg and Y Wang. Nvgre: Network virtualization using generic routing encapsulation. Technical report, 2015.

[5] Dino Farinacci, T Li, S Hanks, D Meyer, and P Traina. Rfc2784: Generic routing encapsulation (gre), 2000.

[6] Google. Google cloud: Cloud computing services. https://cloud.google.com/.

[7] Microsoft. Microsoft azure: Cloud computing services. https://azure.microsoft.com/en-us/.

[8] AT&T. At&t virtual private network. https://www.attgtadirect.com/att-virtual-private-network/.

[9] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast control plane analysis using an abstract representation. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 300–313, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Ari Fogel, Stanley Fung, Luís Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A General Approach to Network Configuration Analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, May 2015.

[11] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 155–168, New York, NY, USA, 2017. Association for Computing Machinery.

[12] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. Tiramisu: fast multilayer network verification. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 201–220, USA, 2020. USENIX Association.

[13] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, Duncheng She, Qing Ma, Biao Cheng, Hui Xu, Ming Zhang, Zhiliang Wang, and Rodrigo Fonseca. 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*, SIGCOMM '20, page 599–614, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Santhosh Prabhu, Kuan-Yen Chou, Ali Kheradmand, P. Brighten Godfrey, and Matthew Caesar. Plankton: scalable network configuration verification through model checking. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 953–968, USA, 2020. USENIX Association.

[15] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. Efficient network reachability analysis using a succinct control plane representation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 217–232, USA, 2016. USENIX Association.

[16] Peng Zhang, Dan Wang, and Aaron Gember-Jacobson. Symbolic router execution. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 336–349, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Ruihan Li, Fangdan Ye, Yifei Yuan, Ruizhen Yang, Bingchuan Tian, Tianchen Guo, Hao Wu, Xiaobo Zhu, Zhongyu Guan, Qing Ma, et al. Reasoning about network traffic load property at production scale. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1063–1082, 2024.

[18] Dan Wang, Peng Zhang, and Aaron Gember-Jacobson. Expresso: Comprehensively reasoning about external routes using symbolic simulation. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 197–212, 2024.

[19] Timothy Alberdingk Thijm, Ryan Beckett, Aarti Gupta, and David Walker. Kirigami, the verifiable art of network cutting. *IEEE/ACM Transactions on Networking*, 2024.

[20] Timothy Alberdingk Thijm, Ryan Beckett, Aarti Gupta, and David Walker. Modular control plane verification via temporal invariants. *Proceedings of the ACM on Programming Languages*, 7(PLDI):50–75, 2023.

[21] Alan Tang, Ryan Beckett, Steven Benaloh, Karthick Jayaraman, Tejas Patil, Todd D. Millstein, and George Varghese. Lightyear: Using modularity to scale BGP control plane verification. In Henning Schulzrinne, Vishal Misra, Eddie Kohler, and David A. Maltz, editors, *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, pages 94–107. ACM, 2023.

[22] Global Times. World's first 1.2tb ultra-high-speed backbone network opens in china. https://www.globaltimes.cn/page/202311/1301765.shtml, 2023.

[23] Nick Giannarakis, Devon Loehr, Ryan Beckett, and David 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*, pages 958–973, 2020.

[24] Xieyang Xu, Yifei Yuan, Zachary Kincaid, Arvind Krishnamurthy, Ratul Mahajan, David Walker, and Ennan Zhai. Relational network verification. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 213–227, 2024.

[25] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. Scalable verification of border gateway protocol configurations with an smt solver. *SIGPLAN Not.*, 51(10):765–780, oct 2016.

[26] Peng Zhang, Aaron Gember-Jacobson, Yueshang Zuo, Yuhao Huang, Xu Liu, and Hao Li. Differential network analysis. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 601–615, 2022.

[27] Peng Zhang, Yuhao Huang, Aaron Gember-Jacobson, Wenbo Shi, Xu Liu, Hongkun Yang, and Zhiqiang Zuo. Incremental network configuration verification. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 81–87, 2020.

[28] Differential datalog (ddlog). https://github.com/vmware/differential-datalog.

[29] Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin Vechev. Bayonet: Probabilistic inference for networks. *ACM SIGPLAN Notices*, 53(4):586–602, June 2018. Funding Information: We thank our shepherd Jean-Baptiste Tristan, as well as Benjamin Bichsel, Keyur Joshi, and the anonymous reviewers for their useful comments on drafts of the paper. This research was supported in part by SNF (grant number 163117) and NSF (Grants No. CCF 1703637 and CCF 1629431). Publisher Copyright: © 2018 ACM.

[30] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In Steven D. Gribble and Dina Katabi, editors, *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 113–126. USENIX Association, 2012.

[31] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: semantic foundations for networks. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, page 113–126, New York, NY, USA, 2014. Association for Computing Machinery.

[32] Nuno P. Lopes, Nikolaj S. Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking beliefs in dynamic networks. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 499–512. USENIX Association, 2015.

[33] Qiao Xiang, Chenyang Huang, Ridi Wen, Yuxin Wang, Xiwen Fan, Zaoxing Liu, Linghe Kong, Dennis Duan, Franck Le, and Wei Sun. Beyond a centralized verifier: Scaling data plane checking via distributed, on-device verification. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 152–166, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide invariants in real time. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, Lombard, IL, April 2013. USENIX Association.

[35] Ryan Beckett and Aarti Gupta. Katra: Realtime verification for multilayer networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 617–634, 2022.

[36] Xu Liu, Peng Zhang, Hao Li, and Wenbing Sun. Modular data plane verification for compositional networks. *Proceedings of the ACM on Networking*, 1(CoNEXT3):1–22, 2023.

[37] Alan Tang, Siva Kesava Reddy Kakarla, Ryan Beckett, Ennan Zhai, Matt Brown, Todd Millstein, Yuval Tamir, and George Varghese. Campion: debugging router configuration differences. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 748–761, 2021.

[38] Siva Kesava Reddy Kakarla, Francis Y Yan, and Ryan Beckett. Diffy: Data-driven bug finding for configurations. *Proceedings of the ACM on Programming Languages*, 8(PLDI):199–222, 2024.

[39] Siva Kesava Reddy Kakarla, Alan Tang, Ryan Beckett, Karthick Jayaraman, Todd Millstein, Yuval Tamir, and George Varghese. Finding network misconfigurations by automatic template inference. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 999–1013, 2020.

[40] Rajdeep Mondal, Alan Tang, Ryan Beckett, Todd Millstein, and George Varghese. What do llms need to synthesize correct router configurations? In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 189–195, 2023.

[41] Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. Netconfeval: Can llms facilitate network configuration? *Proceedings of the ACM on Networking*, 2(CoNEXT2):1–25, 2024.

[42] Lizhao You, Hao Tang, Jiahua Zhang, and Xiao Li. Fast configuration change impact analysis for network overlay data center networks. In *Proceedings of the 4th Asia-Pacific Workshop on Networking*, pages 8–15, 2020.

[43] IPWITHEASE. What is clos architecture or clos network? https://ipwithease.com/clos-architecture/.

[44] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[45] Huawei. https://www.huawei.com/en/.

[46] Z3. https://github.com/Z3Prover/z3.

[47] MEV source code. https://github.com/LXZ230901/MEV.

[48] H3C. Evpn vxlan technology white paper-6w100. https://www.h3c.com/en/Support/Resource_Center/EN/Home/Switches/00-Public/Trending/Technology_White_Papers/EVPN_VXLAN_Technology_WP-6W100/.

[49] Cisco. White papers. https://www.cisco.com/c/en/us/products/switches/nexus-9000-series-switches/white-paper-listing.html.
[50] Huawei. Ip new technologies. https://support.huawei.com/enterprise/en/doc/EDOC1000173015.

A Discussion of correctness

In this section, we discuss the correctness of MEV, which involves two parts: (i) the correctness of ensemble verification in the data plane and (ii) the correctness of ensemble verification in the control plane.

A.1 Correctness of ensemble verification in the data plane

We prove the correctness of ensemble verification for verifying reachability properties in the data plane. Consider a reachability property p, which specifies the forwarding behavior between a source-destination pair $src\text{-}dst_p$ must ensure the packet reaches its destination. During forwarding, the packet may undergo tunneling. We define the following packet forms: (i) pkt_{pre}: original packet before encapsulation, (ii) pkt_{en}: encapsulated packet, and (iii) pkt_{de}: decapsulated packet.

The packet pkt_{pre} originates from VRF vrf_s at router r_s. It is forwarded within network N_1 and matches a rule in the forwarding table of VRF vrf_{t1} at router r_{t1}. At this point, it is encapsulated with a new destination address IP_{en}, producing pkt_{en}. The encapsulated packet pkt_{en} is forwarded through other virtual or physical network N_2 to router r_{t2}, where tunnel endpoint $TE(r_{t2}, dvrf, IP_{en})$ resides. The packet is then decapsulated to pkt_{de} and directed to VRF vrf_{t2} as specified by the encapsulation header. From there, it resumes forwarding within N_1.

The end-to-end forwarding behavior of the packet thus consists of three segments: (i) $behavior_1$: forwarding of pkt_{pre} in N_1 (before encapsulation), (ii) $behavior_2$: forwarding of pkt_{en} in N_2, and (iii) $behavior_3$: forwarding of pkt_{de} in N_1 (after decapsulation). These three behaviors collectively define the packet's forwarding behavior $behavior$ in the real network.

In MEV, the system constructs two nodes in its FDG: node n_1 for the original source-destination pair $src\text{-}dst_p$, and node n_2 for the tunneled source-destination pair $src\text{-}dst_e = (r_{t1}, dvrf, r_{t2}, dvrf, IP_{en})$. At r_{t1}, MEV directly forwards pkt_{pre} to router r_{t2} without modeling the encapsulation step. Therefore, n_1 captures both $behavior_1$ and $behavior_3$. Meanwhile, n_2 is responsible for capturing $behavior_2$ based on a freshly created packet with destination IP_{en}.

We now analyze the different scenarios where p is violated.

- **Case 1:** Violation occurs *before encapsulation*, i.e., $behavior_1$ violates property p. In this case, since pkt_{pre} fails to reach router r_{t1}, the decoupled result for n_1 is False. Consequently, the ensemble result of n_1 is:

$$\mathsf{enResult}(n_1) = \mathsf{False} \wedge (\wedge_{n_i \in n_1.child} \mathsf{enResult}(n_i))$$

  which is always False, indicating a violation of property p.
- **Case 2:** Violation occurs *during the forwarding of encapsulated packet*, i.e., $behavior_2$ violates property p. In this case, the packet pkt_{en} fails to reach router r_{t2} through $N2$. Therefore, n_2's decoupled verification result is False, and:

$$\mathsf{enResult}(n_2) = \mathsf{False} \wedge (\wedge_{n_i \in n_2.child} \mathsf{enResult}(n_i)) = \mathsf{False}$$

  This causes the ensemble result of n_1 to be:

$$\mathsf{enResult}(n_1) = n_1.deResult \wedge \mathsf{False} = \mathsf{False}$$

  which again indicates a violation of p.
- **Case 3:** Violation occurs *after decapsulation*, i.e., $behavior_3$ violates property p. This mirrors Case 1, where the $deResult$ of n_1 is also False.

Therefore, the ensemble verification result $\mathsf{enResult}(n_1) = \mathsf{False}$ if and only if the actual forwarding behavior $behavior$ of $src\text{-}dst_p$ violates the reachability property p.

The correctness for other properties (e.g., loop-freedom, waypointing) can be proved analogously. The main difference lies in the definitions of the check function used to compute $deResult$ and

	No. Verification task	No. Task required repair (No. Repair operation)	No. Inconsistent verification result
MEV	152	-	0
MEV$^+$	152	1 (0)	0

Table 4. Number of verification tasks, tasks requiring repair, and inconsistent verification results.

	Control plane state computation time		Overall verification time	
	Base	Repair	Base	Repair
MEV$^+$	16.762	0.673	105.575	0.961

Table 5. Time of base ensemble verification and repair in MEV$^+$.

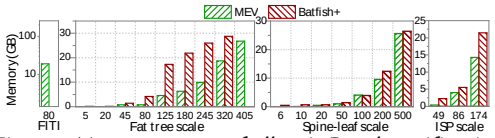


Fig. 11. Memory usage of all-pair Reach verification

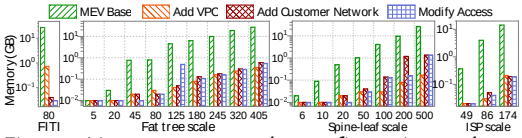


Fig. 12. Memory usage under configuration updates

the enResult function used to compute the ensemble verification result, but the overall proving procedure remains the same. Due to space constraints, we omit these additional proofs. This proving procedure can naturally generalize to the nested encapsulation.

A.2 Correctness of the ensemble verification in the control plane

MEV decouples the dependent conditions between different routing instances or protocols into each individual routing instance and protocol, constructing disjoint process components. For each component, MEV computes its control plan state under the assumption that its dependent conditions hold. In the subsequent verification procedure, based on the network behavior specified by the property to be verified, MEV checks whether the dependent conditions required for corresponding control plane behavior are satisfied, thereby determining whether the control plane behavior holds and ultimately determining whether the property holds.

The control plane states computed by MEV and deployed in the actual network may not always be the same. For example, when a peer relationship is configured between process A and process B, its establishment depends on dep. If there is no error in the network, the routing message will propagate from process A to B and finally reach the destination. However, if dep is not satisfied, in the actual network, the routing message might take an alternative path to the destination. MEV performs the routing propagation assuming dep holds and computes the control plane state of the network. Then, MEV verifies whether dep holds to determine the validity of the current control plane state and does not compute control plane states where dep is not satisfied.

However, this is not a limitation of MEV, due to the errors discovered by MEV do indeed exist and reflect the mismatch between the network configuration and the network operators' intentions. For example, in the case mentioned above, MEV captures the dissatisfaction of dep and reveals that although process A and process B are configured as peers, the peer relationship has not been successfully established. This reflects a discrepancy between the configuration and the operators' intentions, termed as an error. By fixing these discovered configuration errors, network operators can leverage the natural advantage of ensemble verification to quickly re-verify the updated configuration, effectively ensuring the correctness of the network configuration.

Furthermore, as shown in Appendix A.2.1, during the maintenance of FITI, MEV consistently detected all configuration errors without any false positives. Nonetheless, we also developed a more rigorous variant of MEV, called MEV$^+$, and proved its correctness. MEV$^+$ ensures that the control plane state computed through ensemble verification always aligns with the realizable state in the actual network by systematically repairing the FDG. We deployed both MEV and MEV$^+$ on FITI in parallel over a five-month period. Due to space constraints, the details of MEV$^+$ and its formal proofs are included in [47]. Below, we summarize our key observations from deploying both MEV and MEV$^+$.

Category	Type	Examples of Properties
Routing Properties	Existence of Route	Reachability, Isolation, Blocking, Route Leak
	Content of Route	Route Preference, Bounded AS Path, Equal AS Path, Loop-free
Forwarding Properties	Existence of Forwarding Path	Reachability, Isolation, Blocking
	Content of Forwarding Path	Preference, Waypoint, Bounded Forwarding Path, Equal Forwarding Path, Loop-free, Blackhole, Multipath Consistency, Bounded Encapsulation Depth

Table 6. Examples of supported properties

Category	Routers	Injected violations	Identified violations	False alarms
Three-Tier Clos DCNs	5	1	1	0
	20	3	3	0
	45	9	9	0
	80	12	12	0
	180	19	19	0
	405	42	42	0
Two-Tier Clos DCNs	8	1	1	0
	52	10	10	0
	102	19	19	0
	202	31	31	0
	502	49	49	0
ISPs	49	10	10	0
	86	30	30	0
	174	50	50	0

Table 7. Injected violations and MEV detection results

A.2.1 Experience of deploying MEV and MEV$^+$

We deployed both MEV and MEV$^+$ on FITI over a five-month period, conducting 152 verification tasks. As shown in Table 4, only one task required a repair step in MEV$^+$. In all cases, the verification results from MEV and MEV$^+$ were identical. Moreover, MEV$^+$ completed the single repair using just one incremental inference step, with a runtime of less than 1 second (Table 5).

Category	No. Routes	No. Dst.	No. Src-dst pairs	No. Routes
FITI	80	82178	3276800	9442432
Three-Tier Clos DCNs	5	23	12	94
	20	172	600	3726
	45	639	4300	29646
	80	1712	34272	149104
	125	3775	128700	534150
	180	7308	380952	1532844
	245	12887	955500	3761926
	320	21184	2121600	8220096
	405	32967	4290732	16421454
Two-Tier Clos DCNs	8	128	656	1744
	12	212	1932	4544
	22	422	8062	17284
	52	1052	51652	104704
	102	2102	208302	414404
	202	4202	836602	1648804
	502	10502	5241502	10272004
ISPs	49	1142	26100	60256
	86	2893	122500	267611
	174	10777	990000	2809059

Table 8. Destination, route, and src-dst pair counts across networks.

After investigating, we found that our operators configure the peer relationship between different routing processes in the following two situations: (1) Using the IP addresses of the directly connected interfaces, where the peer relationship will definitely be established. Therefore, there is no need to perform the repair operation in MEV$^+$; (2) Using loopback IP addresses, where communication between processes typically depends on other routing instances or protocols. However, in practice, operators usually configure the peer relationships based on the actually reachable loopback interfaces. As a result, repair operations in MEV$^+$ are rarely needed as well.

These findings validate our design choice: although MEV$^+$ provides stronger theoretical guarantees by repairing potential dependency violations, MEV alone already achieves high practical reliability. Our deployment experience demonstrates MEV can serve as a lightweight yet effective solution for continuous network verification in production settings, while MEV$^+$ offers a sound fallback mechanism when stronger assurance is needed.

B More properties

As summarized in Table 6, MEV supports a wide range of routing and forwarding properties. The differences across property verification lie in two aspects: (i) the check function, which computes the

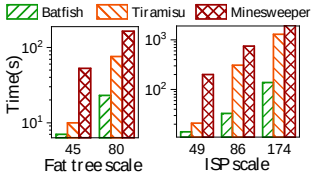


Fig. 13. Verification time without considering link failures

Algorithm 1 Ensemble reasoning

Input: Network topology $topo$, forwarding dependency graph FDG, source-destination pair $src\text{-}dst_p$, converged control plane state $\mathcal{S}^f$, ensemble control plane model N

procedure FIBComputation($topo$, N, $\mathcal{S}^f$)
  for $(r, vrf) \in topo$ **do**
    $V_F \leftarrow$ SelectVertices(r, vrf, N)
    $rib \leftarrow$ GetConnectOrStaticRoutes(r, vrf)
    for $v \in V_F$ **do**
      MergeMessage(rib,$\mathcal{S}^f(C_i)(v)$)
    end for
    $fib \leftarrow$ ConvertMessage(rib)
    SetFIB(r, vrf,fib)
  end for
end procedure

procedure Fwd($topo$, $src\text{-}dst(r_s, vrf_s, r_d, vrf_d, prefix_d)$)
  $pkt \leftarrow$ InitializePkt($src\text{-}dst$)
  InjectPkt(r_s, pkt)
  $unConverge \leftarrow$ True
  while $unConverge$ **do**
    $unConverge \leftarrow$ False
    for $r \in topo$ **do**
      for $pkt \in$ GetPkt(r) **do**
        $vrf \leftarrow$ MatchVRF(r, pkt)
        $FIB \leftarrow$ GetFIB(r, vrf)
        $f \leftarrow$ MatchFIB(FIB, pkt)
        $pkt.P$.add(r, vrf)
        $pkt.D$.add($f.D$)
        ForwardToNextHop($topo$, f, pkt)
      end for
    end for
    for $r \in topo$ **do**
      if GetPkt(r) $\neq \emptyset$ **then**
        $unConverge \leftarrow$ True
      end if
    end for
  end while
end procedure

procedure EnsembleReasoning($topo$, $src\text{-}dst_p$, $\mathcal{S}^f$, FDG, N)
  FIBComputation($topo$, N, $\mathcal{S}^f$)
  $FDG.unReason \leftarrow \emptyset$
  $n \leftarrow$ CreateNode($src\text{-}dst_p$)
  AddNode(FDG, n)
  while $FDG.unReason \neq \emptyset$ **do**
    for $n_u \in FDG.unReason$ **do**
      Fwd($topo$, $n_u.src\text{-}dst$)
      $pkt_r \leftarrow$ GetRecvPkt($topo$, $src\text{-}dst_p$)
      UpdateFDG(FDG, pkt_r)
    end for
  end while
end procedure

decoupled verification result for each FDG node, and (ii) the enResult function, which computes the ensemble verification result for each FDG node. Due to space limitations, we describe the detailed procedures for verifying each property in our technical report [47].

C Extra Experiment Results

As shown in Table 7, we systematically injected diverse reachability violations into synthetic networks, based on real-world misconfigurations observed in collaboration with the NoC of FITI and public reports from technical forums [48] and configuration guides [49, 50]. In all cases, MEV successfully detected every violation with zero false positives.

Figure 11 reports MEV's memory usage when verifying all-pairs reachability. Batfish$^+$ timed out on FITI and fat-tree networks with 405 routers, so memory data is unavailable for those cases. Figure 12 presents MEV's memory usage during incremental verification under configuration updates. Table 8 summarizes the number of destinations, total routes in the converged control plane, and the number of source-destination pairs across different networks. Figure 13 shows the verification time of Batfish, Tiramisu, and Minesweeper when checking all-pairs reachability on various ISP and data center networks without considering link failures.

D Algorithms

Algorithm 1 illustrates the procedure of FIB computation, packet forwarding, and ensemble reasoning for forwarding behavior.

Received June 2024; accepted September 2025