HifiCNet: High-fidelity Cloud Network Validation Platform at Scale by Hybrid Architecture

Jiawei Liu[1,2], Gongming Zhao[1,2], Hongli Xu[1,2,3], Baoqing Wang[1,2], Peng Yang[1,2], Chun-Jen Chung[4], Min Chen[5], Xuwei Yang[5]

[1] School of Computer Science and Technology, University of Science and Technology of China
[2] Suzhou Institute for Advanced Research, University of Science and Technology of China
[3] Hefei National Laboratory, University of Science and Technology of China
[4] Huawei Technologies Canada Co., Ltd. [5] Huawei Technology Co., Ltd.
liujiawei@mail.ustc.edu.cn,{gmzhao,xuhongli}@ustc.edu.cn, {bbq,ypholic}@mail.ustc.edu.cn,
{james.chung,chenmin148,yangxuwei1}@huawei.com

***Abstract*—Ensuring reliable operation of cloud networks is critical for cloud service providers to guarantee quality of service for tenants. A promising solution is to design a high-fidelity cloud network validation platform that proactively validates the correctness of all operations before implementing changes to the production network. However, the tight coupling between physical and virtual networks in the cloud poses challenges to achieving high-fidelity cloud network validation. Existing network validation platforms focus primarily on traditional physical networks, while ignoring virtual network validation. Regrettably, neglecting the combined validation of physical and virtual networks will result in inaccurate evaluations. To bridge this gap, we present HifiCNet, a high-fidelity platform that concurrently validates both physical and virtual networks. HifiCNet designs an orchestrator to elegantly coordinate the interaction between physical and virtual networks in the cloud and innovatively adopts an emulator-simulator hybrid architecture to ensure high fidelity and scalability for cloud network validation. Through extensive evaluation based on real topologies and traffic traces, we show that HifiCNet enables high-fidelity validation of cloud network configurations, services, and exceptions. Notably, HifiCNet can use 38 servers to establish a physical network comprising 10k hosts, and a virtual network consisting of 200k virtual machines.**

Index Terms*— *Cloud Network, Network Validation, Physical Network, Virtual Network.

I. Introduction

Cloud computing has emerged as the dominant paradigm for delivering a wide range of services, such as data storage, high-performance computing, and artificial intelligence [1]. As a crucial infrastructure of cloud computing, the cloud network facilitates communication among various cloud components and services, empowering tenants to deploy and access applications [2]. Ensuring the reliable operation of the cloud network is of utmost importance for cloud service providers (CSPs), such as Amazon EC2 [3] and Alibaba Cloud [4].

However, it is far from trivial to maintain a reliable cloud network environment due to its vast and complex characteristics. Even minor issues such as misconfiguration, human error, or equipment failure will result in significant network congestion and service disruption [2, 5, 6]. For example, misconfigured routing entries in a virtual private cloud (VPC) may create a loop and cause traffic disruption (see details in §VII-C2). To address this challenge, a promising solution is to design a high-fidelity cloud network validation platform that proactively validates the correctness of all operations before implementing changes to production networks [7].

Achieving high-fidelity validation of cloud networks requires a thorough understanding of their features. Unlike traditional networks, cloud networks consist of tightly coupled physical and virtual networks (*i.e.*, underlay and overlay networks) [8, 9, 10]. Specifically, the physical network encompasses the underlying infrastructure of the cloud, including routers, switches, servers, and ethernet cables within the cloud [11]. It is managed by the CSP and remains transparent to tenants. In contrast, virtual networks are built on top of the physical network, enabling connectivity between virtual machines (VMs), virtual network functions (VNFs), and other cloud resources [12]. Virtual networks provide an abstraction layer that allows tenants to create and configure VPCs based on their specific requirements. Thus, the characteristics and conditions of virtual networks depend on the physical network, while the performance of physical networks is closely related to the business of virtual networks [9]. Neglecting either aspect compromises the reliability of cloud network validation. For example, access control lists (ACLs) in the virtual network and switch table entries in the physical network both influence traffic routing. Neglecting either aspect will lead to distorted validation results of traffic routing. Therefore, a high-fidelity cloud network validation platform must conduct co-emulation of both physical and virtual networks.

It is a pity that existing network validation platforms [7, 13, 14, 15, 16, 17] mainly focus on traditional physical networks and neglect virtual network validation in the cloud. For instance, both NS3 [13] and Mininet [16] are good at rapidly deploying and validating physical networks with minimal resource costs. However, when it comes to validating cloud networks, NS3 lacks virtual network modeling, leading to validation distortion [18], while Mininet cannot emulate the virtual networks due to insufficient node functionality and isolation [7]. Although a few studies acknowledge the importance of joint validation of virtual and physical networks [19, 20],

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

Type		Emulator			Simulator		Hybird
Platforms		Cloudlab [19] Emulab [21]	Mininet[16]	Distrinet [17] CrystalNet[7]	NS3[13] / NS4[14] OMNet++[15]	CloudSim [20]	Ours
Fidelity	Physical Network	●	●	●	◑	○	●
	Virtual Network	●	○	○	○	○	●
Scalability	Physical Network	○	○	◑	●	●	●
	Virtual Network	○	○	○	○	●	●

Notes: ● = Good, ◑ = Fair, ○ = Bad.

TABLE I: Comparison of Network Validation Platforms. Existing platforms can be categorized into emulators and simulators, with most of them primarily focusing on building physical networks and paying less attention to validating virtual networks. Emulators excel in fidelity but have poor scalability, while simulators excel in scalability but suffer from fidelity.

they struggle to achieve high-fidelity validation at scale. For example, CloudLab [19] employs an emulation-based solution, using specific infrastructure to achieve high-fidelity validation of both physical and virtual networks. However, this method consumes a substantial amount of resources, limiting its ability to validate large-scale cloud networks. Conversely, CloudSim [20] adopts a simulation-based approach, utilizing a discrete event simulator (DES) to model both physical and virtual networks. While it enables large-scale validation, it falls short in accurately simulating real-world network traffic, leading to a decrease in fidelity.

To bridge this gap, we aim to design a high-fidelity cloud network validation platform at scale. However, designing such a platform presents significant challenges. Firstly, it is essential to conduct a joint validation of both physical and virtual networks for high-fidelity. However, considering the tight coupling between physical and virtual networks in the cloud, emulating their interactions is difficult. Secondly, the high-fidelity joint validation entails the consumption of considerable resources, which in turn affects the platform scalability. To conquer the above challenges, this paper proposes HifiCNet and highlights its main contributions as follows:

- We emphasize the importance of jointly validating physical and virtual networks to ensure reliable cloud network operations, analyze existing solution limitations, and conclude the design goals for the cloud network validation platform.
- We design HifiCNet, a high-fidelity cloud network validation platform at scale. It features an orchestrator to coordinate interactions between physical and virtual networks in clouds, and adopts an emulator-simulator hybrid architecture to balance fidelity and scalability.
- We evaluate the performance of HifiCNet by building a cloud network validation platform with 38 servers. This platform can replicate the production environment topology, creating a physical network with 10k hosts and a virtual network with 200k VMs. It enables high-fidelity validation across various cloud scenarios, including network configurations, service deployments, and exception handling.

II. Background and Motivation

A. Scenarios of Cloud Network Validation

Scenario 1: Configuration Validation. Cloud networks involve various configurations, including router and switch configurations in the physical network, as well as VPC and VNF configurations in the virtual network [22]. These configurations undergo frequent adjustments driven by business requirements or performance optimization needs. However, the interdependencies among configurations pose a challenge in determining the optimal configuration and validating its availability. Inappropriate configurations will lead to decreased application performance or errors [23]. For example, updating the ACLs in a VPC will impact the routing paths in the cloud, and incorrect ACL configurations can result in network disruptions [24]. To avoid large-scale network disruptions, comprehensive validation of new configurations is essential before deploying them in production [7].

Scenario 2: Service Validation. Cloud networks host various services from tenants. However, specific changes to services, such as upgrades or migrations, will pose irreversible impacts on the network [25]. For instance, during the migration of a distributed Voice over Internet Protocol (VoIP) service, traffic and load are transferred from one node to another node. However, employing improper migration strategies will lead to network congestion, resulting in service degradation (see details in §VII-C2). Since service changes often involve modifications to network configurations, or routing, accurately assessing their expected results poses a significant challenge.

Scenario 3: Exception Validation. Exceptions such as malicious attacks and device failures are prevalent in clouds [26]. For CSPs, it is essential to investigate the robustness of cloud networks in the presence of anomalous conditions, using a high-fidelity validation platform. This approach helps identify potential weaknesses within the cloud, enabling timely remediation measures to mitigate the impact of similar anomalous events in production environments. For example, understanding the impact of distributed denial-of-service (DDoS) attacks with varying intensities on tenants is crucial for informing the design of additional security measures [27].

Although these scenarios have distinct focuses, they share two common features. Firstly, cloud networks encompass both physical and virtual networks, making it essential to consider the interactions and dependencies between them. Neglecting either aspect will result in incomplete or inaccurate validation outcomes. Therefore, *the joint validation of physical and virtual networks is imperative*. Secondly, cloud networks are known for their vast scale, *e.g.*, a Google data center housing 10k hosts and 100k VMs [28]. Recent research emphasizes the importance of conducting network experiments at a scale close to real deployment to maintain confidence in the results [29]. Conducting comprehensive validation of all details in the entire network will consume a significant amount of resources and be time-consuming. Therefore, *accomplishing large-scale*

validation with limited infrastructure resources is crucial.

B. Limitations of Prior Works

To validate the correctness of operations before implementing changes to the production network, *emulators have shown potential by utilizing infrastructure or virtualization techniques to emulate network devices* [7, 16, 17, 19, 21]. CloudLab [19] and EmuLab [21] leverage their own infrastructure to construct networks, resulting in highly faithful emulation of physical and virtual networks in the cloud. However, these methods have poor scalability due to cost and resource constraints (see details in §VII-B). Alternatively, emulators like Mininet [16], Distrinet [17], and CrystalNet [7] utilize various virtualization technologies to emulate network devices. For example, Mininet utilizes namespaces to emulate network devices, enabling the emulation of networks with hundreds of hosts and switches on a single server [16]. However, Mininet lacks support for distributed deployment and has limitations in functionality and isolation, impeding its ability to emulate virtual networks. Distrinet [17] and CrystalNet [7] overcome these issues by leveraging Linux container daemon (LXD) [30] and utilizing VMs inside Docker containers, respectively, thereby facilitating distributed deployment. Nonetheless, these emulators primarily focus on emulating physical networks and can not effectively replicate the complex behaviors of virtual networks in the cloud. Furthermore, while virtualization-based emulators exhibit improved scalability compared to infrastructure-based approaches, they still face scalability challenges when emulating large-scale cloud networks.

Unlike emulators, *simulators utilize CPU computing power to abstract network resources into mathematical models and simulate network behavior, achieving high scalability* [13, 14, 15, 20]. However, due to simplified models and assumptions that may not align with actual network behavior, simulators often suffer inaccuracies and produce unrealistic results. For instance, NS3 [13], a popular open-source network simulator, is renowned for modeling physical networks, while NS4 [14] effectively models programmable networks by extending NS3. OMNeT++ [15] performs well in modeling distributed networks. Nonetheless, these simulators encounter model distortion when simulating cloud networks due to the lack of virtual network modeling in the cloud. CloudSim [20] aims to bridge this gap by modeling both physical and virtual networks using a DES. However, it has limited support for modeling complex real-world network behaviors, such as network congestion, and cannot accurately simulate traffic in the cloud.

Table I summarizes the pros and cons of existing network validation platforms. Existing works still have limitations on cloud network validation. There is an urgent need for *a high-fidelity, scalable cloud network validation platform that enables joint validation of physical and virtual networks*.

C. Our Intuition

As discussed in §II-B, emulators have the potential to provide high-fidelity replicas of physical and virtual networks, but their scalability is poor. On the other hand, simulators excel in scalability but often sacrifice fidelity due to the use of simplified models. This raises the question: *Is it possible to combine the advantages of emulators and simulators for cloud network validation?* In fact, different validation scenarios exhibit diverse fidelity requirements for distinct parts of the cloud network [7]. For example, when evaluating the impact of various service migration strategies (see details in §VII-C2), it is crucial to monitor the traffic of network components (*e.g.*, hosts, VMs, *etc.*) within the rack where the service is deployed, as well as the traffic of components in the potential target rack for migration. Therefore, using an emulator scheme is a suitable choice for this specific part of the network components. For other parts of the cloud, the focus is primarily on observing whether the migration configuration has any unexpected effects or not. Considering the large number of components in those parts, the simulator method is preferred.

Based on this insight, we propose *a novel solution called emulator-simulator hybrid architecture*. Specifically, cloud network components that require focused attention in a validation scenario will be validated using an emulator. For other components that do not necessitate extensive attention to detail, a simulator can be employed to enhance scalability. By effectively coordinating components in the emulator and simulator, we can leverage their strengths and achieve high-fidelity, large-scale cloud network validation.

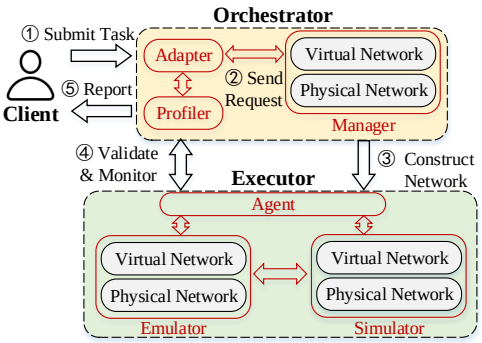


Fig. 1: HifiCNet Overview and Workflow. Client provides APIs for users. Orchestrator manages the interactions between physical and virtual networks. Executor performs the actual validation.

III. System Design

A. Design Goals

High-fidelity. HifiCNet possesses the ability to accurately replicate the behavior of actual production networks, including both physical and virtual networks, as well as their intricate interactions. Additionally, it should be capable of effectively validating common network events in the cloud.

Scalability. HifiCNet empowers data center-scale validation of both physical (*e.g.*, 10k hosts) and virtual networks (*e.g.*, 100k VMs) while utilizing limited resources.

B. System Overview

HifiCNet adopts an innovative emulator-simulator hybrid architecture, comprising three key modules: Client, Orchestrator and Executor, as shown in Fig. 1.

Client acts as the user interface, providing HifiCNet's API (§III-D) for users to submit tasks, such as topology adjustment, VPC creation, *etc.* It also allows users to retrieve network status information such as latency, bandwidth, *etc.*

Orchestrator elegantly coordinates physical and virtual networks within cloud environments while collecting network state information. It consists of three submodules: *Adapter*, *Manager*, and *Profiler*. Adapter (§IV-A) receives the user-submitted tasks, and decomposes them into specific requests that can be handled by Manager and Profiler components. Manager (§IV-B) responds to these requests, and distributes specific instructions for managing the physical network and virtual network to Executor. Profiler (§IV-C) allows users to monitor resource state information and proactively validate the network by injecting network events.

Executor handles network construction and scenario validation based on Orchestrator's management. It consists of three submodules: *Agent*, *Emulator*, and *Simulator*. The Agent (§V-A) builds the cloud network by controlling the Emulator and Simulator as per Orchestrator's instructions. The Emulator (§V-B) replicates the production environment's topology, configuration, applications, and traffic with high fidelity, accurately mimicking the complex behaviors and interactions of virtual and physical networks. The Simulator (§V-C) enhances network validation scalability by using simplified applications and traffic models while maintaining consistent topology and configuration with the production environment.

C. Workflow

Fig. 1 also outlines the HifiCNet system workflow, which includes 5 steps: ① Users submit network validation tasks to the Orchestrator via the Client API, leveraging their observations of the production network and specific validation scenario requirements. ② The Adapter converts these tasks into requests for the Manager and Profiler. ③ The Manager analyzes the requests and sends instructions to the Agent in Executors, which sequentially constructs the cloud network within the Emulator and Simulator modules. ④ After building the cloud network, users can utilize the Profiler to inject network events, such as burst traffic and node failures, for validation. The Profiler monitors the impact of these events, collecting and organizing performance data. ⑤ Users can evaluate the network by accessing validation reports organized by the Profiler.

D. HifiCNet APIs

HifiCNet provides a user-friendly and unified API through the Client, categorized into three types, based on the validation workflows in §III-C. The first type manages physical network components, enabling operations such as adding, deleting, setting, and displaying hosts, switches, routers, and links within physical networks and cloud environments. The second type focuses on managing virtual network components, facilitating similar operations for VPCs, subnets, VNFs, VMs, and applications running within VMs. Adapters are crucial in translating these API calls into requests directed at network

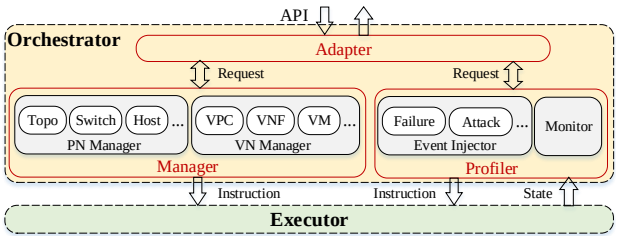


Fig. 2: Orchestrator Design. Adapter decomposes tasks into specific requests. Manager distributes concrete instructions to Executors. Profiler injects network events and monitors the network state.

managers. The third type is specifically designed for the Profiler, which allows users to inject configurations, generate traffic, and introduce exceptions into the network. It also provides monitoring capabilities, including setting parameters, displaying information, and logging data to observe the impact on network performance and behavior.

IV. Orchestrator Design

A. Adapter Design

The Adapter receives tasks described by users via the API from the Client and converts them into specific requests, according to predefined rules that can be handled by other submodules. Additionally, when users acquire network status information, the Adapter extracts data collected by the Profiler and presents it to the Client.

B. Manager Design

As shown in Fig. 2, the Manager consists of the Physical Network (PN) Manager and the Virtual Network (VN) Manager. It handles requests from the Adapter and makes decisions to manage both networks and their interactions.

Physical Network Management. To validate physical networks at scale, distributed construction across multiple servers is essential. HifiCNet tackles this challenge with the PN Agent (§V-A), deployed on each server and registered with the PN Manager. This setup allows servers to construct the physical network based on the PN Manager's instructions. During construction, the PN Manager assigns PN nodes, such as hosts and switches, to appropriate servers. Users can customize the deployment strategy of those nodes; by default, the PN Manager uses the weight round robin (WRR) algorithm [31] to ensure load balancing among servers.

Virtual Network Management. The VN Manager manages the virtual network, including VPCs, VNFs, and VMs, through pre-deployed VN Agents (§V-A) on each emulated host in the physical network. OpenStack [32], known for its scalability and rich features, is the chosen solution for this task. We utilize OpenStack's Keystone and Neutron for VPC and VNF management, while Nova handles VM lifecycle management. To support ultra-large-scale virtual networks, we deploy OpenStack in a distributed manner, utilizing distributed message queues and database clusters to boost the VN Manager's scalability and performance [33, 34].

Interaction Management. The characteristics of virtual networks are influenced by the underlying physical network.

Therefore, HifiCNet adopts a stepwise construction approach: first, the physical network is fully constructed to ensure proper configuration; then, the virtual network is built on top, aligning with the physical network. Additionally, the order of operations within the virtual network can impact the physical network. To maintain the desired sequence, the Manager establishes an asynchronous message queue [35] with each Agent. This allows the Manager to send instructions to the queue based on the task, ensuring operations are processed by the Agents in the correct order.

C. Profiler Design

The Profiler comprises two main components: the Event Injector and the Monitor. The Event Injector facilitates proactive validation by injecting events like configuration updates, burst traffic, and network exceptions into the Executor. Initially, it requests target node information from the Manager and translates user API-described tasks into executable instructions for the nodes. For example (see §VII-C2), when the Event Injector receives a packet corruption task for the emulated switch *eSW1* via the API call *inject('eSW1', 'exception', 'corrupt', 'port1', '0.5', '100s')*, it requests the corresponding PN agent ID from the PM Manager. The Event Injector then translates this task into a command sequence executable by eSW1. Specifically, it uses traffic control (TC) [36] and netem [37] tools to control *port1*, causing a 0.5% probability of random packet corruption for 100 seconds:

```
tc qdisc add dev port1 root netem delay 10s
tc qdisc change dev port1 root netem corrupt 0.5%
sleep 100
tc qdisc del dev port1 root netem
```

These instructions are sent to the corresponding agents via an asynchronous message queue. The node then receives these instructions from its agent and executes them sequentially, completing the event injection process. Meanwhile, the Monitor uses observation tools like ping and top to periodically collect network status information. It allows configuring key performance indicators, such as latency, enabling timely detection of potential issues during validation.

V. Executor Design

A. Agent Design

The agent bridges the Orchestrator and Executor, facilitating distributed cloud network validation across various infrastructures. It consists of two components: the PN Agent and the VN Agent. The PN Agent, deployed as a daemon on the server, receives instructions for the physical network from the Orchestrator and constructs the network by controlling the Emulator and Simulator. In contrast, the VN Agent, pre-installed as a daemon on each emulated host, receives instructions for the virtual network from the Orchestrator and establishes the virtual network through its control over the Emulator and Simulator.

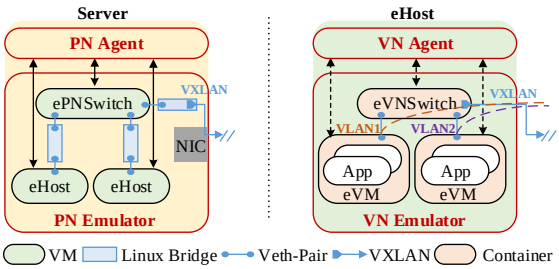


Fig. 3: Emulator Design. (a) *Left plot:* PN Emulator uses VMs to emulate eHosts and ePNSwitches; (b) *right plot:* VN Emulator leverages containers to emulate eVMs and eVNSwitches.

B. Emulator Design

1) Physical Network Emulator: **Emulated PN Node.** To accurately replicate the physical network, the PN Emulator uses virtual machines (VMs) for node emulation, each distinguished by different VM images, as illustrated in the left plot of Fig. 3. It categorizes PN nodes into three main types: emulated hosts (eHosts), emulated switches (ePNSwitches), and emulated routers (ePNRouters). Specifically, the eHost is equipped with the VN Agent for virtual network construction. The ePNSwitches use an image with Open vSwitch (OVS), allowing flexible configuration and management of flow entries to control data flow behavior. The ePNRouters utilize an image that includes Border Gateway Protocol (BGP) or Open Shortest Path First (OSPF). Additionally, VMs offer flexibility in emulating heterogeneous devices by adjusting configurations such as memory and CPU.

Emulated PN Link. HifiCNet features two types of physical network links: data plane links and management plane links. Data plane links (blue links in the left plot of Fig. 3) are treated as Ethernet links by devices and must be isolated from one another. For each physical network link, the PN Emulator establishes a Linux Bridge and utilizes the TC [36] tool to configure bandwidth, packet loss rates, and delay on the link. Two sets of veth pairs are created on the Linux Bridge to connect the devices at each end of the link. Notably, in scenarios where high-bandwidth links (*e.g.*, 400Gbps) need to be emulated on lower bandwidth hardware (*e.g.*, 10Gbps), HifiCNet employs time dilation techniques [38, 39]. Additionally, data plane links must have the ability to span across multiple underlying infrastructures to construct distributed network topologies. To address this requirement, the PN Emulator establishes Virtual eXtensible LAN (VXLAN) tunnels between ePNSwitches. VXLAN is chosen over other tunneling protocols, such as GRE because it best meets our objectives. It emulates Ethernet links, and the outer header (UDP) allows us to connect across any IP network, including the wide-area Internet [7]. On the other hand, management plane links (black links in the left plot of Fig. 3) use additional veth pairs to connect all PN nodes to the Agent on a server. These links perform two main functions: (1) they aid the Orchestrator in updating the management plane configuration, such as modifying flow entries in the ePNSwitches; (2) they assist the Orchestrator in collecting status information from nodes during the validation process.

2) Virtual Network Emulator: **Emulated VN Node.** Considering that the nested VM approach may compromise the platform stability [40], we use containers to emulate VN nodes, as shown in the right plot of Fig. 3. Containers offer a lightweight virtualization technology that not only ensures platform stability but also significantly enhances the scalability of the VN Emulator (see details in §VII-B). Similar to PN nodes, different nodes can be distinguished by various container images. For example, we can run emulated VMs (eVMs) with the same applications as in the real world.

Emulated VN Link. Similar to the PN Emulator, the VN Emulator also categorizes its links into data plane and management plane links. Data plane links (blue links in Fig. 3) within the same eHost are established using veth-pairs, ensuring direct and efficient communication between VN nodes on the same eHost. For connectivity between VN nodes across different eHosts, the VN Emulator establishes VXLAN tunnels over the PN Emulator's data plane links. Considering the presence of multiple VPCs within the virtual network, it becomes crucial to ensure isolation between them. To address this, the VN Emulator utilizes virtual local area networks (VLANs) to differentiate and segregate the various VPCs. On the other hand, management plane links (black links in Fig. 3) play a crucial role in coordinating updates to the management plane configuration and collecting node status information.

C. Simulator Design

1) Design Principles: Balancing scalability and fidelity is the core principle of our Simulator's design. Traditional simulators, as discussed in II-B, use mathematical and discrete event models to replicate network behavior. However, these simplified models often result in inaccurate and unrealistic outcomes. A distorted, large-scale simulator holds no value for network validation. The main challenge causing model distortion is the complexity of network configuration and management-data plane interactions [41, 42]. To address this, we enhance our Simulator's fidelity by integrating a lightweight management plane emulator, based on two key insights. First, the management plane is the only entity that can alter network behavior, making its fidelity crucial. Second, management plane messages are far fewer than data plane traffic [41], minimizing their impact on the Simulator's scalability.

2) Detailed Design: Following these principles, our lightweight management plane emulator aims to accurately represent network configuration and management-data plane interactions within the Simulator, without delving into the specifics of data plane traffic. To achieve this, we utilize customized lightweight containers to construct the physical network in the cloud and employ namespaces to customize the virtual network. Notably, these nodes are stripped of other functionalities, retaining only the ability to respond to management plane configurations, thereby minimizing the impact on the Simulator's scalability.

Message Receiver. The next challenge is synchronizing the management plane configurations and interactions with the Simulator model. The key lies in the forwarding entries within

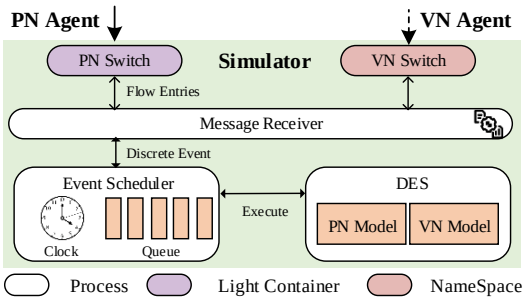


Fig. 4: Simulator Design. Message Receiver collects messages and converts them into discrete events. Event Scheduler maintains a event Queue and Clock. DES maintains network models.

switches and routing entries in routers. In both physical and virtual networks, modifications in the management plane are ultimately transformed into these entries. The specific forwarding behavior of data plane traffic strictly depends on them. To facilitate this synchronization, Message Receivers act as crucial bridges. They convert these entries into simulated discrete events, which are then inserted into the Event Scheduler for model synchronization. This process is triggered by specific situations: (1) during the initialization of the Simulator, (2) when entry updates occur in switches and routers, and (3) through scheduled polling by the Message Receiver.

Event Scheduler. The sequence of event processing in network validation can lead to vastly different outcomes. To ensure discrete events are handled as expected, the Event Scheduler plays a crucial role. Upon receiving events from the Message Receiver, the Event Scheduler assigns each an execution timestamp based on its arrival time and priority, organizing them into an event queue. Additionally, the Event Scheduler maintains an event execution clock. When an event's timestamp matches the current clock time, it is dequeued and forwarded to the DES for execution, ensuring the flow and accuracy of event processing.

Discrete Event Simulator. DES is responsible for managing and maintaining both the physical and virtual network models. Upon the arrival of discrete events, DES utilizes customizable handler functions to process these events and update the network models accordingly. These handler functions allow users to modify the physical and virtual network models in response to specific discrete events. For instance, if a link failure event occurs, DES disables the corresponding link in the model to accurately represent the network's state.

VI. Discussion and Implementation

A. Discussion

Role Handoff. HifiCNet supports dynamic role switching, allowing nodes to transition between simulated and emulated modes without a system-wide restart, inspired by live VM migration in cloud computing [43, 44]. To switch from a simulated to an emulated node, the Orchestrator instructs the Executor to create a new emulated node and gradually transfer the configuration data. Once complete, the simulated node is temporarily shut down, the emulated node is brought online, registration is updated, and traffic is migrated. The emulated

node then undergoes a behavioral check, and if it runs reliably, the simulated node is removed. Switching from an emulated node to a simulated node follows a similar process.

Node Role Determination. As HifiCNet is designed for cloud network developers and operators, we currently allow users to manually determine node roles based on validation scenario requirements and the following guidelines, with plans for automatic determination in the future: (1) Opt for emulated nodes when resources are ample and fidelity is crucial; choose simulated nodes when resources are limited and scalability is prioritized. (2) Assign nodes requiring high-fidelity data plane validation as emulated nodes. (3) If uncertain, users can start with a simulated node and switch to an emulated node as needed during validation.

Interaction between Simulated and Emulated Nodes. During network validation, simulated nodes excel at identifying potential network configuration anomalies, while emulated nodes are adept at traffic validation. It is recommended to implement both nodes as emulated if frequent traffic interaction exchange is expected. In the few cases where traffic interaction is required between emulated and simulated nodes, HifiCNet uses the Message Receiver to analyze traffic from emulated nodes and convert it into simulated discrete events based on packet headers, with each packet's analysis and conversion taking approximately 2µs, making the overhead negligible. These events are then queued in chronological order by the Event Scheduler. The DES processes these events sequentially using its network model and returns results to the Message Receiver, which then uses these results to generate traffic for the emulated nodes to complete the interaction.

Trade-offs and Weaknesses. HifiCNet is designed for scenarios requiring the joint validation of a large number of nodes in both physical and virtual networks, with a subset running actual production applications. These scenarios primarily arise in the later stages of network design and during the operational phase, as discussed in §II. In contrast, simulator-based validation platforms are better suited for the early stages of network design, focusing on rapid large-scale prototyping and model validation. Additionally, while HifiCNet can emulate device behavior and assess the impact of faults such as packet loss and corruption (§VII-C2), it is not designed for verifying hardware design issues (*e.g.*, ASIC-related). HifiCNet is not suited for detecting bugs caused by gradual state accumulation (*e.g.*, memory leaks) or timing-sensitive issues (*e.g.*, multi-thread race conditions). For such purposes, validation platforms with physical infrastructure are more appropriate.

B. Implementation

We developed HifiCNet using OpenStack [32] and MiniNet [16], totaling approximately 10k lines of code. The Client, Adapter, and Profiler are all implemented in Python, with 715, 815, and 543 lines of code, respectively. The PN Manager (1.1k lines) and PN Agent (745 lines) are both implemented in C++. The VN Manager is built according to OpenStack Yoga [32], while the VN Agent follows nova-compute and neutron-openvswitch-agent [32]. As for the Executor, within the Emulator, we deploy the PN Agent on each server to emulate the physical network. Similar to Distrinet [17], we re-implement host control (1.6k lines), link control (577 lines), switch control (305 lines), and network construction (1k lines) using Python, based on Mininet 2.3 [16]. This enables distributed physical network construction using VMs or Docker containers. To emulate the virtual network, we pre-package the VN Agent and OVS 2.17 [45] in PN node images. We also re-implemented OpenStack's fake driver (1.3k lines) in Python to construct virtual networks using lightweight containers and namespaces. In the Simulator, we implement the Message Receiver (770 lines) in Python, the event scheduler based on [41], and the DES (1.3k lines) in C++.

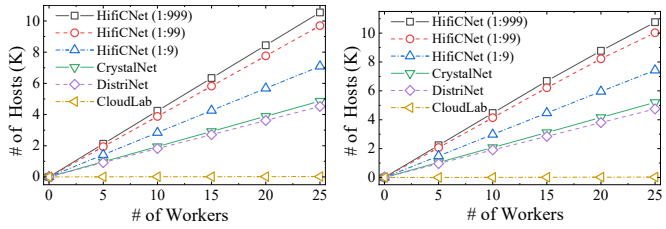
VII. Evaluation

A. Experimental Settings

Setup. All experiments are conducted using 1 personal computer (PC) and 38 servers, all running Ubuntu 22.04 [46]. The PC, equipped with an 8-core CPU and 32GB of memory, serves as the client, remotely accessing the server cluster via SSH. Each server features dual 22-core Intel Xeon 6161 CPUs, 640GB of memory, and an Intel XL710 40GbE NIC. The servers are interconnected via a Huawei CloudEngine 6885-SAN-56F switch with 40Gbps links. Among the servers, 12 are designated for the Orchestrator cluster, while the remaining 26 are allocated for the Executor cluster. In the Orchestrator cluster, 11 servers are dedicated to the distributed deployment of the Manager, with the last server handling other components. In the Executor cluster, the Message Receiver, Event Scheduler, and DES are deployed on one server. The remaining 25 servers run other components, named *"workers"* for clarity. For workers, by default, customized VMs with 4 vCPUs and 8GB memory are used to construct the emulated physical network. Docker containers with Ubuntu 22.04 are used to construct the emulated virtual network. To ensure model fidelity in the DES, customized lightweight Docker containers and namespaces are employed in the Simulator.

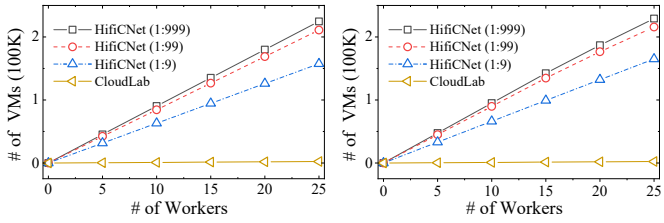
Performance Metrics. We give an outline of the following two sets of performance metrics. Firstly, to evaluate the scalability of HifiCNet, we measure the *maximum number of hosts* in a constructed physical network and the *maximum number of VMs* in a constructed virtual network, with the increasing number of workers. Additionally, we demonstrate the time cost of building physical and virtual networks by measuring the *completion time* for building data centers (DCs) and VPCs with varying sizes. Secondly, to assess the fidelity, Table II compares the validation capabilities of seven typical cloud events. We also observe various network performance metrics through case studies, including *network throughput, flow completion time (FCT), packet loss rate, requests per second (RPS)* and *round trip time (RTT)*, in §VII-C.

Benchmarks. HifiCNet adopts an innovative emulator-simulator hybrid architecture, positioning it as a comprehensive platform for data center-scale validation. Given that many common network validation scenarios in the cloud focus on a small subset of nodes within the data center, typically



(a) Fat-Tree (b) Leaf-Spine

Fig. 5: Scale of Physical Network



(a) Fat-Tree (b) Leaf-Spine

Fig. 6: Scale of Virtual Network

less than 10% of the total number of nodes [7], we assess various ratios of emulated and simulated nodes: 1:999, 1:99, and 1:9. These configurations are referred to as *HifiCNet (1:999)*, *HifiCNet (1:99)*, and *HifiCNet (1:9)*, respectively. In addition, we compare the performance of HifiCNet with three other network validation platforms. First, *CrystalNet* [7], a virtualization-based emulator by Microsoft, runs applications or device VMs inside Docker containers on cloud VMs, focusing on configuration, software bugs, and human failures in the cloud's physical network. The second benchmark is *Distrinet* [17], a distributed software-defined network emulator based on virtualization technology, implemented on Mininet with LXD containers to enhance node isolation and functionality. The third is *CloudLab* [19], an emulator developed by Utah University, allowing users to construct small-scale cloud networks on its infrastructure with high fidelity. Notably, we exclude simulators like NS3 as benchmarks since they use simplified mathematical models for simulation, which cannot accurately capture complex network behavior and performance characteristics in the cloud, as discussed in §II-B.

B. Evaluation on Scalability

Scale of Physical Network. We first investigate the maximum number of hosts that our platform can accommodate in the physical network by varying the number of workers. Notably, we solely consider the physical network topology in this set of experiments, without initiating any virtual services on the hosts. We examine two widely used data center topologies in the cloud: fat-tree topology and leaf-spine topology [47]. The results in Fig. 5 showcase the outstanding scalability of our platform in terms of physical network construction. For instance, as shown in Fig. 5(b), with 25 workers in the leaf-spine topology, HifiCNet(1:999), HifiCNet(1:99), and HifiCNet(1:9) successfully emulate 10,756, 10,025, and 7,430 hosts, respectively. In comparison, CrystalNet supports 5,700 hosts, DistriNet handles 4,200 hosts, and CloudLab is limited to 25 hosts. Our platform's advantage lies in its innovative emulator-simulator hybrid solution. Furthermore, as virtualization increases—from CloudLab's direct infrastructure approach to Distrinet's LXD solution and CrystalNet's use of Docker containers—scalability improves, but network quality correspondingly declines.

Scale of Virtual Network. Secondly, we focus on the scalability of the virtual network. To account for device heterogeneity, we randomly create 10-50 VMs [48, 49] in each host. By varying the number of workers, we observe the maximum number of VMs created in the virtual network, as shown in Fig. 6. The results highlight the excellent scalability of our platform in terms of virtual network construction. For example, in the leaf-spine topology with 25 workers, HifiCNet(1:999), HifiCNet(1:99), and HifiCNet(1:9) successfully emulate 229,103, 215,685, and 164,946 VMs, respectively. In comparison, CloudLab achieves 2,200 VMs. This advantage is derived from the utilization of container or namespace technologies in our hybrid architecture, in contrast to CloudLab's reliance on KVM for VM creation. Notably, neither DistriNet nor CrystalNet can emulate virtual networks.

Time Cost. Due to space limitations, we only provide a conclusion. For physical network creation, HifiCNet, Distrinet, and CrystalNet have similar average completion times, while CloudLab takes significantly longer. Constructing a small data center (S-DC) with 25 hosts in a leaf-spine topology takes 4.9, 5.2, 6.8, and 36.7 minutes for HifiCNet, Distrinet, CrystalNet, and CloudLab, respectively. This is because CloudLab builds the physical network from scratch, unlike the other methods that use more efficient virtualization technologies. For virtual network creation, HifiCNet completes the task in half the time of CloudLab, while Distrinet and CrystalNet cannot construct virtual networks. Building a virtual network with 2,200 virtual machines in the S-DC takes HifiCNet 18.5 minutes, compared to CloudLab's 61.2 minutes. The time difference is due to CloudLab's reliance on KVM for VM creation, whereas HifiCNet uses containers or namespaces for rapid VM emulation.

C. Evaluation on Fidelity

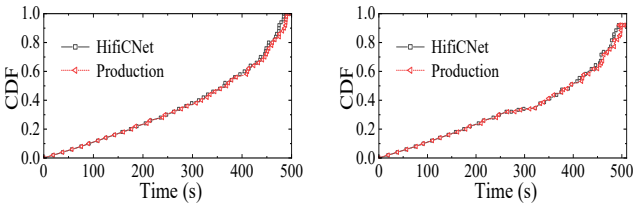
To demonstrate our platform's fidelity in network verification, we first compare HifiCNet's ability to verify common network events. Then, we explore its capabilities in-depth through three typical application scenarios in production environments.

1) Network Event Validation: We conducted a comprehensive comparison of HifiCNet with other benchmarks to evaluate their capabilities in common network event validation, as summarized in Table II. Due to space constraints, we select three aspects for detailed introduction: ① *PN Infrastructure Deployment:* This includes hosts, links, switches, routers, and network topology customization. CloudLab is limited by its infrastructure and cannot customize device capabilities, whereas other platforms fully support this aspect. ④ *VPC Configuration:* This includes network, subnet, security group, VM, and ACL configurations. Distrinet and CrystalNet lack support, while CloudLab and HifiCNet fully support these configurations. ⑦ *Network Attacks:* CloudLab lacks support,

Network Event Type		Examples	DistriNet[17]	CrystalNet[7]	CloudLab[19]	Ours
PN	① Infra. Deployment	infrastructure capability, topology customization	●	●	◑	●
	② Infra. Conf.	routing policy, switch entry, link delay/packet loss	●	◑	◑	●
VN	③ VNF Deployment	load balancer, firewall, DHCP customization	○	○	●	●
	④ VPC Conf.	network, subnet, security group, ACL	○	○	●	●
PN & VN	⑤ Comp. Failure	fiber cuts, power failures, VM outages, VNF failures	◑	○	◑	●
	⑥ Traffic Injection	traffic injection in infra. or VPCs	◑	◑	◑	●
	⑦ Network Attacks	DDoS, DNS attacks against infra. or VPCs	◑	◑	○	●

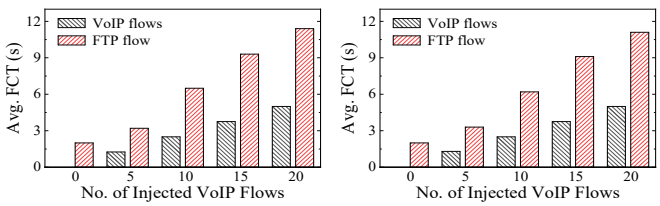
Notes: Infra. = Infrastructure, Conf. = Configuration, Comp. = component ● = Supported, ◑ = Partially Supported, ○ = Not Supported.

TABLE II: Comparison of Network Event Validation Capabilities between HifiCNet and other Benchmarks



(a) Configuration1 (b) Configuration2

Fig. 7: Configuration Validation



(a) HifiCNet (b) Production

Fig. 8: Service Deployment

Distrinet and CrystalNet only support the physical network. HifiCNet offers comprehensive support.

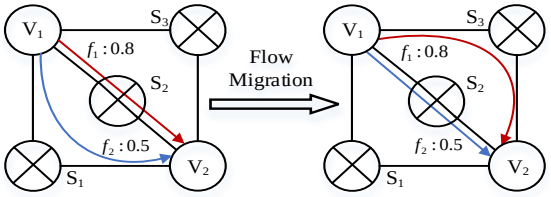
2) Case Study: To evaluate fidelity, we select three typical production network validation scenarios, as discussed in §II-A. We construct a physical network within HifiCNet, modeled after a small data center in our production environment. This network includes 2k hosts arranged in a leaf-spine topology with 1 Gbps link bandwidth between nodes. We then build a VPC comprising 10k VMs and 20 subnets [50] in both the physical network of the data center and HifiCNet for subsequent experiments, referring to the former as *"Production"*.

2.1) Configuration Validation: In this scenario, our focus is to establish interconnectivity between the subnets within the VPC by deploying different configurations, rather than evaluating traffic performance. Therefore, we use *HifiCNet (1:999)* to validate this scenario. Initially, the 20 subnets in the VPC are isolated. We deploy two configurations, *Configuration1* and *Configuration2*, to enable connectivity. *Configuration1* is the correct setup from the production environment, while *Configuration2* modifies parts of *Configuration1* to create loops between some subnets. To validate the correctness of the configuration deployment, we randomly select one VM from each subnet and perform connectivity tests using the ping tool. After deploying the configurations, we calculate the Cumulative Distribution Function (CDF) for the moments when VMs can successfully communicate with each other, as shown in Fig. 7. The results indicate that HifiCNet and Production yield consistent outcomes. In *Configuration1*, the CDF progressively increases over time until it reaches 1, indicating successful communication among all VMs and validating its correctness. However, in *Configuration2*, the CDF temporarily stops growing around 270s, but resumes afterward without reaching 1. These findings demonstrate that our platform can *validate the correctness of configurations and assist in identifying configuration errors*.

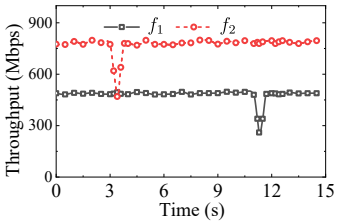
2.2) Service Validation: As discussed in §II-C, in this scenario, the CSP needs to conduct detailed traffic observations of the cloud components involved in the service during validation. For other parts of the network, the focus is on observing any unexpected impacts caused by migration configurations. Therefore, we employ *HifiCNet (1:99)* to validate this scenario. We employ *HifiCNet (1:99)* to validate this scenario.

Service Deployment. We investigate the impact of introducing a new service on the existing services, with an example of adding a VoIP service to the existing FTP service. In the physical network, we randomly select two leaf switches, S_1 and S_2, connected by a link L_1, and under each switch, a host, H_1 and H_2. In the virtual network, we select two emulated VMs from each host, labeled V_{1-1}, V_{1-2}, V_{2-1}, and V_{2-2}. Initially, we transfer 10 files, each 20 MB, between V_{1-1} and V_{2-1} using FTP and record the FCT. Next, while the transfer continues, we introduce 5-20 VoIP flows, each 50 MB, between V_{1-2} and V_{2-2}. Our results, shown in Fig. 8, indicate that in HifiCNet, the FTP flow's FCT increases from 2s to 11.4s, while the VoIP flows' average FCT rises from 1.25s to 5.1s. In Production, the FTP flow's FCT goes from 2s to 11.1s, and the VoIP flows' average FCT from 1.3s to 5s. This increase is due to the shared link L_1 becoming a bandwidth bottleneck. These findings demonstrate our platform's high-fidelity capture of *the impact of new services on existing services during deployment*.

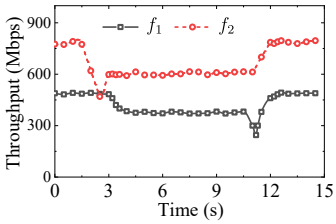
Service Migration. In this case, we investigate the effects of different migration strategies on tenant services. In the physical network, we randomly select hosts H_1 and H_2 from different leaf switches, connected through spine switches S_1-S_3, as shown in Fig. 9(a). In the virtual network, we randomly choose VMs V_1 and V_2 located on H_1 and H_2, respectively. Two VoIP service flows, f_1 and f_2, with bandwidths of 0.8 Gbps and 0.5 Gbps, are established. The goal is to migrate the service traffic from the current scheme to the target scheme. We consider two migration strategies: *Strategy1* involves migrating f_1 first, followed by f_2, while *Strategy2* involves migrating f_2 first, followed by f_1. Using HifiCNet, we evaluate the performance of these strategies, as depicted in Fig. 9(b) and 9(c). Results indicate that *Strategy1* has a minor impact on tenant services, whereas *Strategy2* has



(a) Example: migrate the traffic from schemes (1) to (2)

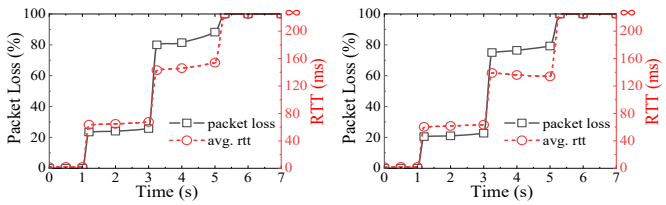


(b) *Strategy1*: $f1$ first, then $f2$



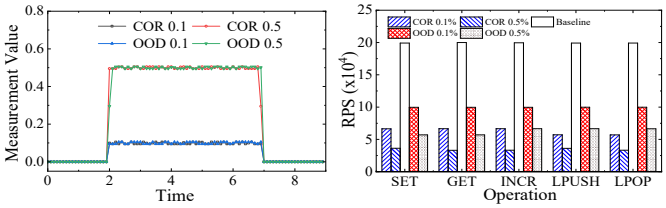
(c) *Strategy2*: $f2$ first, then $f1$

Fig. 9: Service Migration



(a) HifiCNet (b) Production

Fig. 10: DDoS Attack



(a) Injection Validation (b) RPS under Switch Fault

Fig. 11: Equipment Faulty

a more significant impact due to potential link congestion during migration. Our experiments in Production yield similar results, but due to space constraints, they are omitted here. These findings underscore the significance of our platform in *evaluating performance discrepancies among migration strategies during service migration*.

2.3) Exception Validation: In this scenario, to comprehensively evaluate exceptions, we use *HifiCNet (1:9)* for validation.

DDoS Attack. We investigate the impact of DDoS attacks on tenant service. Initially, we measure the packet loss rate and RTT between two emulated VMs, V_1 and V_2, within a VPC under normal conditions, as shown in Fig. 10. Next, we randomly select three additional emulated VMs within the same VPC and sequentially launch DoS attacks, such as UDP floods, targeting V_2. Following these attacks, we measure the packet loss rate and RTT between V_1 and V_2 to assess the impact of the DDoS attacks. The results indicate that as the number of attackers increases, tenant service performance gradually deteriorates, eventually leading to service interruption. Note that the experiments in Production yield similar results. This demonstrates that HifiCNet can *validate the impact of network attacks in the cloud on tenant service*.

Equipment Faulty. In this case, we focus on switch packet corruption (COR) and out-of-order delivery (OOD) faults within a distributed Redis cluster. Following best deployment practices [51], we set up a Redis cluster with 3 primary nodes, 3 backup nodes, and 3 sentinel nodes on emulated VMs, with data exchange routed through a Leaf switch labeled S_1. We first validate its capability to create COR and OOD events. as shown in Fig. 11(a), we transmit a test flow between a primary and a backup node, deliberately introducing COR and OOD packets into the egress port of S_1 for 5 seconds at rates of 0.1% and 0.5%, respectively. The results confirm that our platform can successfully emulate these switch faults. Next, we establish a performance baseline by measuring the requests per second (RPS) for various operations under normal network conditions. We then reintroduce COR and OOD packets into the egress traffic of S_1 at the same rates and measure the RPS again, as illustrated in Fig. 11(b). The results demonstrate the significant impact even minor fault rates can have on performance; for instance, a mere 0.1% rate of packet COR leads to a substantial 66.6% decrease in RPS for the INCR operation. Notably, similar results were observed in production environments, though detailed data is omitted here due to space constraints. These findings clearly illustrate HifiCNet's capability to *accurately assess the impact of equipment faults in the cloud on tenant services*.

VIII. Conclusion

In this paper, we introduce HifiCNet, a high-fidelity cloud network validation platform that jointly validates physical and virtual networks through an innovative emulator-simulator hybrid architecture. Experimental results demonstrate that HifiCNet can use 38 servers to construct a physical network with 10k hosts and a virtual network with 200k VMs. Furthermore, HifiCNet enables high-fidelity validation across various cloud scenarios, such as network configuration, service deployment, and exception handling. In the future, we aim to extend HifiCNet to incorporate new technologies, such as RDMA and programmable switches, to support a wider range of scenarios.

Acknowledgement

Gongming Zhao and Hongli Xu are the co-corresponding authors. We thank our shepherd Chen Tian and anonymous reviewers for their insightful comments. The authors from USTC are supported in part by the National Science Foundation of China (NSFC) under Grants 62372426, 62132019, and 62102392; the Fundamental Research Funds for the Central Universities; the Innovation Program for Quantum Science and Technology under grant 2021ZD0300705; and the Youth Innovation Promotion Association of the Chinese Academy of Science (2023481).

References

[1] B. Magotra, D. Malhotra, and A. K. Dogra, "Adaptive computational solutions to energy efficiency in cloud computing environment using vm consolidation," *Archives of Computational Methods in Engineering*, pp. 1–30, 2022.

[2] G. Zhao, J. Liu, Y. Zhai, H. Xu, and H. He, "Alleviating the impact of abnormal events through multi-constrained vm placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1508–1523, 2023.

[3] Amazon ec2. https://docs.aws.amazon.com/ec2/index.html. Accessed: April. 10, 2024.

[4] Alibaba cloud. https://us.alibabacloud.com. Accessed: April. 10, 2024.

[5] Google. google compute engine incident no.16007. connectivity issues in all regions. https://status.cloud.google.com/incident/compute/16007. Accessed: June. 10, 2023.

[6] J. Wang, G. Zhao, H. Xu, Y. Zhai, Q. Zhang, H. Huang, and Y. Yang, "A robust service mapping scheme for multi-tenant clouds," *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1146–1161, 2022.

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

[8] S. Sekigawa, C. Sasaki, and A. Tagami, "Toward a cloud-native telecom infrastructure: Analysis and evaluations of kubernetes networking," in *2022 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2022, pp. 838–843.

[9] What is an overlay network. https://info.support.huawei.com/info-finder/encyclopedia/en/Overlay+network.html. Accessed: April. 10, 2024.

[10] Cloud networking: Next-generation data center networks. https://dgtlinfra.com/cloud-networking-data-center-networks/. Accessed: June. 10, 2023.

[11] L. Luo, G. Zhao, H. Xu, L. Xie, and Y. Xiong, "Vita: Virtual network topology-aware southbound message delivery in clouds," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 630–639.

[12] C. Fang, H. Liu, M. Miao, J. Ye, L. Wang, W. Zhang, D. Kang, B. Lyv, P. Cheng, and J. Chen, "Vtrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 31–43.

[13] Ns-3 network simulator. https://www.nsnam.org/. Accessed: June. 10, 2023.

[14] J. Bai, J. Bi, P. Kuang, C. Fan, Y. Zhou, and C. Zhang, "Ns4: Enabling programmable data plane simulation," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–7.

[15] Omnet++: Discrete event simulator. https://omnetpp.org/. Accessed: April. 10, 2024.

[16] Mininet. http://mininet.org/. Accessed: June. 10, 2023.

[17] G. Di Lena, A. Tomassilli, D. Saucez, F. Giroire, T. Turletti, and C. Lac, "Distrinet: A mininet implementation for the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 51, no. 1, pp. 2–9, 2021.

[18] J. Cao, Y. Liu, Y. Zhou, L. He, and M. Xu, "Turbonet: Faithfully emulating networks with programmable switches," *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1395–1409, 2022.

[19] Cloudlab. https://www.cloudlab.us/. Accessed: April. 10, 2024.

[20] Cloudsim. http://cloudbus.org/cloudsim/. Accessed: June. 10, 2023.

[21] Emulab. https://www.emulab.net/. Accessed: April. 10, 2024.

[22] B. Chen, H. Yao, H. Chen, Y. Wang, B. Shen, C. Shen, and P. Yang, "An automatic configuration framework for cloud-network infrastructure," in *2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 6. IEEE, 2023, pp. 1695–1699.

[23] C. Zhao, T. Chugh, J. Min, M. Liu, and A. Krishnamurthy, "Dremel: Adaptive configuration tuning of rocksdb kv-store," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 2, pp. 1–30, 2022.

[24] M. Ibrar, L. Wang, G.-M. Muntean, A. Akbar, N. Shah, and K. R. Malik, "Prepass-flow: A machine learning based technique to minimize acl policy violation due to links failure in hybrid sdn," *Computer Networks*, vol. 184, p. 107706, 2021.

[25] G. Boulougaris and K. Kolomvatsos, "An inference mechanism for proactive service migration at the edge," *IEEE Transactions on Network and Service Management*, 2023.

[26] K. N. Qureshi, G. Jeon, and F. Piccialli, "Anomaly detection and trust authority in artificial intelligence and cloud computing," *Computer Networks*, vol. 184, p. 107647, 2021.

[27] M. A. S. Monge, A. H. González, B. L. Fernández, D. M. Vidal, G. R. García, and J. M. Vidal, "Traffic-flow analysis for source-side ddos recognition on 5g environments," *Journal of Network and Computer Applications*, vol. 136, pp. 114–131, 2019.

[28] M. Fadaeefath Abadi, F. Haghighat, and F. Nasiri, "Data center maintenance: applications and future research directions," *Facilities*, vol. 38, no. 9/10, pp. 691–714, 2020.

[29] R. Jansen, J. Newsome, and R. Wails, "Co-opting linux processes for {High-Performance} network simulation," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 327–350.

[30] Lxd:linuxcontainers. https://linuxcontainers.org/lxd. Accessed: April. 10, 2024.

[31] W. Wang and G. Casale, "Evaluating weighted round robin load balancing for cloud web services," in *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE, 2014, pp. 393–400.

[32] Openstack's 25th release, yoga, harmonizes with hardware, cloud-native tools; retires technical debt to promote well-being of stable, reliable core. https://www.openstack.org/software/yoga/. Accessed: April. 10, 2024.

[33] D. Haja, M. Szabo, M. Szalay, A. Nagy, A. Kern, L. Toka, and B. Sonkoly, "How to orchestrate a distributed openstack," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 293–298.

[34] How we scaled openstack to launch 168,000 cloud instances. https://superuser.openinfra.dev/articles/how-we-scaled-openstack-to-launch-168-000-cloud-instances/. Accessed: April. 10, 2024.

[35] Rabbitmq is the most widely deployed open source message broker. https://www.rabbitmq.com/. Accessed: April. 10, 2024.

[36] tc - show / manipulate traffic control settings. https://man7.org/linux/man-pages/man8/tc.8.html. Accessed: April. 10, 2024.

[37] netem - netem provides network emulation functionality for testing protocols by emulating the properties of wide area networks. https://wiki.linuxfoundation.org/networking/netem. Accessed: April. 10, 2024.

[38] H. W. Lee, D. Thuente, and M. L. Sichitiu, "Integrated simulation and emulation using adaptive time dilation," in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2014, pp. 167–178.

[39] H. W. Lee, M. L. Sichitiu, and D. Thuente, "Neat: Network link emulation with adaptive time dilation," *Journal of Parallel and Distributed Computing*, vol. 104, pp. 88–98, 2017.

[40] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th international middleware conference*, 2016, pp.

1–13.
[41] E. L. Fernandes, G. Antichi, I. Castro, and S. Uhlig, "An sdn-inspired model for faster network experimentation," in *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2018, pp. 29–32.
[42] D. Jin and D. M. Nicol, "Parallel simulation of software defined networks," in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2013, pp. 91–102.
[43] D. Fernando, P. Yang, and H. Lu, "Sdn-based order-aware live migration of virtual machines," in *IEEE INFOCOM 2020-IEEE conference on computer communications*. IEEE, 2020, pp. 1818–1827.
[44] T. He and R. Buyya, "A taxonomy of live migration management in cloud computing," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–33, 2023.
[45] Production quality, multilayer open virtual switch. https://www.openvswitch.org/. Accessed: June. 10, 2023.
[46] Ubuntu 22.04.2 lts (jammy jellyfish). https://releases.ubuntu.com/jammy/. Accessed: April. 10, 2024.
[47] Z. Liu, Y. Zhao, Z. Fan, T. Yang, X. Li, R. Zhang, K. Yang, Z. Jiang, Z. Zhong, Y. Huang, C. Liu, J. Hu, G. Xie, and B. Cui, "Burstbalancer: Do less, better balance for large-scale data center traffic," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–18, 2023.
[48] A. Roy, S. Midya, K. Majumder, and S. Phadikar, "Distributed resource management in dew based edge to cloud computing ecosystem: A hybrid adaptive evolutionary approach," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 8, p. e4018, 2020.
[49] M. Lakzaei, V. Sattari-Naeini, A. Sabbagh Molahosseini, and A. Javadpour, "A joint computational and resource allocation model for fast parallel data processing in fog computing," *The Journal of Supercomputing*, vol. 78, no. 10, pp. 12 662–12 685, 2022.
[50] J. Sun, T. Wo, X. Liu, T. Ma, X. Mou, J. Lan, N. Zhang, and J. Niu, "Scalable inter-domain network virtualization," *Journal of Network and Computer Applications*, vol. 218, p. 103701, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804523001200
[51] V. Kylmämaa, "Horizontal scalability and high availability of a hospital information system," Master's thesis, V. Kylmämaa, 2023.