Cloud Load Balancers Need to Stay Off the Data Path

Yuchen Zhang*, Shuai Jin†, Zhenyu Wen‡, Shibo He*, Qingzheng Hou†, Yang Song†, Zhigang Zong†, Xiaomin Wu†, Bengbeng Xue†, Chenghao Sun†, Ku Li†, Xing Li†, Biao Lyu*†, Rong Wen†, Jiming Chen*, *Fellow, IEEE*, Shunmin Zhu†

*Zhejiang University, Hangzhou, Zhejiang, China
†Alibaba Group, Hangzhou, Zhejiang, China
‡Zhejiang University of Technology, Hangzhou, Zhejiang, China

***Abstract*—Load balancers (LBs) are crucial in cloud environments, ensuring workload scalability. They route packets destined for a service (identified by a virtual IP address, or VIP) to a group of servers designated to deliver that service, each with its direct IP address (DIP). Consequently, LBs significantly impact the performance of cloud services and the experience of tenants. Many academic studies focus on specific issues such as designing new load balancing algorithms and developing hardware load balancing devices to enhance the LB's performance, reliability, and scalability. However, we believe this approach is not ideal for cloud data centers for the following reasons: (i) the increasing demands of users and the variety of cloud service types turn the LB into a bottleneck; and (ii) continually adding machines or upgrading hardware devices can incur substantial costs.**

In this paper, we propose the Next Generation Load Balancer (NGLB), designed to bypass the TCP connection datapath from the LB, thereby eliminating latency overheads and scalability bottlenecks of traditional cloud LBs. The LB only participates in the TCP connection establishment phase. The three key features of our design are: (i) the introduction of an *active address learning* model to redirect traffic and bypass the LB, (ii) a *multi-tenant isolation* mechanism for deployment within multi-tenant Virtual Private Cloud networks, and (iii) a distributed flow control method, known as *hierarchical connection cleaner*, designed to ensure the availability of backend resources. The evaluation results demonstrate that NGLB reduces latency by 16% and increases nearly 3× throughput. With the same LB resources, NGLB improves 10× rate of new connection establishment. More importantly, five years of operational experience has proven NGLB's stability for high-bandwidth services.

***Index Terms*—Cloud Computing, Load Balancer**

I. Introduction

The Load Balancer (LB) is an essential component of cloud service infrastructure, as indicated by numerous studies [1]–[8]. It efficiently allocates client requests across backend servers, ensuring optimal resource use and balanced processing of requests [9]–[12]. The role of LB as the intermediary in Application-Cloud network interactions critically influences the quality of cloud services [13]–[15]. As cloud applications scale and the demand for network resources grows, the performance requirements for LBs are increasingly stringent [15]. Concurrently, advancements in application microservices and the consolidation of computing resources in cloud networks significantly heighten the demands on LBs. Previous research reports that LBs manage nearly half of all data center traffic [5], [16].

Most studies have primarily concentrated on enhancing scheduling algorithms [17]–[22] or improving the hardware performance of LBs [12], [13]. Several innovative studies have introduced the Direct Server Return (DSR) mode [23], utilizing Full Network Address Translation (FNAT). For instance, Maglev [1], developed by Google, employs DSR to send packets directly back to routers, thereby bypassing the need for Maglev to handle the returning packets and effectively halving the load on the LB. DSR remains insufficient for services that require high bidirectional bandwidth, such as storage solutions, or latency-sensitive applications, such as online gaming. Moreover, the centralized nature of current load balancing solutions tends to have traffic bottlenecks, rendering them unsuitable for applications demanding high throughput and low latency.

The crux in cloud services is the commitment to stability and real-time responsiveness. Based on our experience, the prevailing centralized traffic control strategy has turned the LB into a significant bottleneck within the traffic of cloud data centers, resulting in the following issues.

Problem 1: Excessive resource consumption and delay caused by high traffic passing through LB. This problem arises from the ongoing expansion of Alibaba Cloud's operations, which results in a steady increase in north-south traffic. Figure 1 shows the traffic dynamics within a single Alibaba Cloud cluster over 30 days, where the average throughput surpasses 1.8 Tbps and has been consistently increasing at a

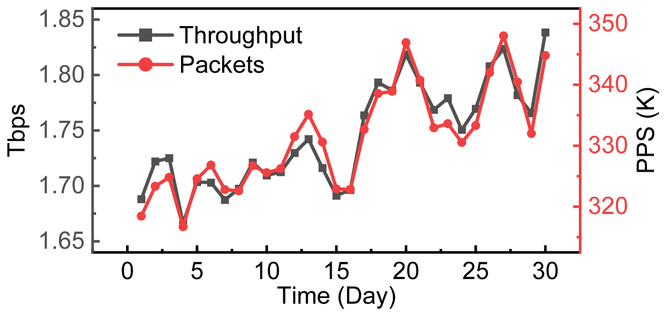


Fig. 1: A Single Cluster Traffic Growth

* Shibo He is the corresponding author.
* Shunmin Zhu is the corresponding author.

rate of about 2% per month. Under the current centralized load balancing approach, all data packets are required to go through the LB for processing and forwarding. This setup significantly intensifies the issues during periods of high data traffic, leading to a sharp increase in resource usage.

Given the limitations imposed by the breakdown of Moore's Law, simply boosting LB hardware performance or refining scheduling algorithms offers only temporary solutions. Therefore, cloud service providers must reduce the volume of traffic passing through LBs, possibly by bypassing them in the traffic flow.

Problem 2: Poor availability of LB in cloud services. The LB serves as an essential link within the traffic network, with its performance significantly impacting the accessibility of all operational cloud services. Any disruptions or modifications to the LB may lead to interruptions in newly established connections, owing to delays in session synchronization [24].

To reduce the impact of LB disruptions on cloud services, Alibaba Cloud employs a dual-zone strategy that incorporates both a primary and a backup availability zone. In the event of a failure in the primary zone, the system smoothly transitions to the backup zone, maintaining uninterrupted service through a reliable fail-safe mechanism. Furthermore, Alibaba Cloud implements a master-standby setup, where the primary availability zone for one device also serves as the backup for another. It's crucial to keep the resource utilization of each LB below 50% to ensure smooth service continuation after a switch. However, this approach necessitates a doubling of resource use and costs associated with the LBs, leading to decreased resource efficiency and adding complexity to the cloud service architecture.

Problem 3: The bottleneck of LB scalability. To manage the surge in traffic effectively, LBs must scale rapidly to meet changing service demands. However, creating a responsive and precise auto-scaling system poses significant challenges. Scaling rules need to be established based on application performance metrics. Rules that are too sensitive can lead to frequent scaling actions, incurring unnecessary costs and potentially destabilizing the system [25], [26]. On the other hand, rules that lack sensitivity may result in delayed responses, failing to address demand promptly [27], [28].

Additionally, within data centers, LBs encounter scaling limitations due to Per-Connection Consistency (PCC) rules [12], [22]. These rules mandate that traffic from the same user must be directed to the same server, requiring a mechanism for session synchronization. This increases system complexity, consumes substantial network bandwidth, and can ultimately restrict LB scalability.

The aforementioned issues stem from centralized traffic management strategies. To address this, Alibaba Cloud has introduced Next Generation Load Balancer (NGLB), which is designed to support high-bandwidth, low-latency, and scalable cloud solutions. This system minimizes the LB's role to only handling initial connection requests (SYN packets) and staying out of the ongoing data path. NGLB's innovative approach involves semi-distributed traffic management to overcome traditional challenges. Next, we will outline the three key designs of the NGLB.

First, to address the challenges associated with cross-plane communication after decoupling the LB from the data path, we introduce an innovative address learning technique that enables traffic to bypass the LB effectively. This approach fosters seamless, direct interactions between clients and servers by refining an *active address learning* mechanism, distinctively functioning across clients, LB, and servers. The client assimilates the destination address from direct traffic and updates its session table accordingly. Utilizing TCP options provided by the LB, the server identifies client details and updates the server-side session table accordingly (see §IV).

Secondly, to adapt to the inherently multi-tenant nature of cloud services, we have developed an *address translation* method. Recognizing that different users function within separate address spaces, it is crucial for cloud service providers to prevent address overlaps during user access. To this end, we utilize Virtual eXtensible Local Area Network (VXLAN) [29] technology to separate various tenants, while transferring the address translation responsibilities from the LB to the server. This strategy efficiently prevents address conflicts that could occur when multiple tenants access cloud services using the same addresses (see §V).

Third, to optimize backend server resource utilization, we devise an innovative flow management strategy. Considering that NGLB is designed for high-bandwidth applications, maximizing backend resource efficiency is of utmost importance. Our approach circumvents the occupation of these resources by vast idle flows. By distributing the flow management function from the LB to the servers, we deploy a simplified version of the TCP protocol stack on each server. Terminating idle connections prevents an excessive number of idle connections from occupying backend servers and thus averting performance degradation.

Experiments demonstrate that NGLB outperforms a product-level LB currently in use in our data center, by delivering a throughput that's 3 times greater and reducing 16% latency. Finally, we summarize our experience in deploying NGLB (§VIII) and raise several open research questions on the design of a more mature NGLB.

Therefore, this paper makes the following contributions:

- We introduce NGLB, an L4 software LB specifically designed for cloud networks. With the innovative proactive address learning mechanism, it bypasses LB at the data level, reducing latency by 16% and increasing 3 $\times$ throughput.
- We design a new multi-tenant access mechanism that does not depend on LB, ensuring that multiple tenants can maintain access to cloud services simultaneously. This mechanism ensures that different tenants can have independent address spaces on the client side, and using any address within this space to access cloud services will not result in address conflicts.
- We introduce a new traffic clean-up method to ensure the stability of high-traffic services using NGLB. Fine-grained timeout settings and a bidirectional active disconnection scheme allow idle traffic to be cleaned quickly, freeing up backend resources.

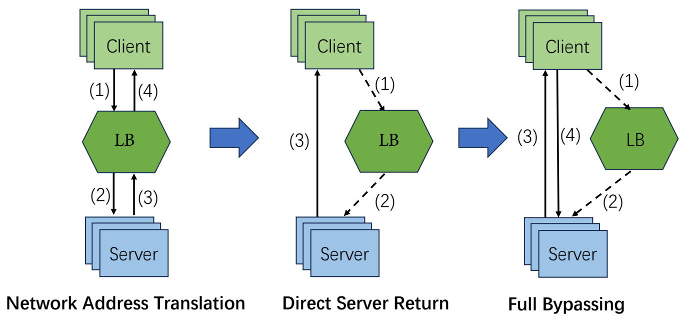


Fig. 2: The Evolution of L4 Load Balancer

II. Motivation and Background

Alibaba Cloud Load Balancer (ALB) was initially designed for layer-4 load balancing within the Alibaba Cloud ecosystem, accessible through both traditional and Virtual Private Cloud (VPC) networks. Leveraging FNAT, ALB provides a solid foundation for traffic distribution. However, as cloud service scale has evolved, expectations for load balancing solutions have grown. To address this, Alibaba Cloud introduced ALB Destination Network Address Translation (ALB-DNAT), an optimized forwarding mode based on DSR. This enhanced the capabilities of the original ALB. With the ongoing expansion and increasing diversity of cloud services, continuous optimization has become essential, especially for managing bidirectional traffic within VPC networks. This need led to the development of NGLB, the latest advancement in Alibaba Cloud's VPC network load balancing solutions. This work explores the evolution of Alibaba Cloud's load balancing modes, highlighting the addressed challenges and the innovations introduced in NGLB.

A. The role of LB in Alibaba Cloud

Alibaba Cloud data centers employ the LB to support a variety of services, such as the Cloud Relational Database Service (RDS), SQL Server, and the Cloud Object Storage Service (OSS). By deploying across multiple availability zones, the LB ensures the uptime of LB instances exceeds 99.95%. The LB improves the efficiency and availability of backend services by virtualizing multiple cloud servers within the same region into a robust service pool. It distributes client requests to these servers based on predefined forwarding rules. Simultaneously, the LB monitors the health of the servers in the pool and automatically isolates those with abnormal statuses, thereby mitigating single-point failures and enhancing overall application performance. Moreover, the LB is fortified against DDoS attacks, significantly enhancing the security of application services.

B. The evolution of Alibaba cloud LB

As shown in Figure 2, current load balancing modes can be divided into three schemes: FNAT [30]–[32], DSR [5], [33], [34], and Full Bypassing [35], [36].

ALB was introduced over ten years ago. This FNAT-based LB was designed to evenly distribute network traffic evenly, marking the beginning of rapid development in cloud networking. However, ALB faces significant issues in high-download scenarios such as OSS. To address this, we optimized the ALB forwarding mode, creating an ALB-DNAT forwarding mode based on DSR. Unlike ALB, ALB-DNAT directs server response traffic straight to the client, bypassing the LB. This approach alleviates the bandwidth strain on the LB and allows around 80% of the traffic to bypass it.

Despite these improvements, ALB-DNAT still has several issues similar to those in ALB. First, the benefits of DSR are limited to server-originated outbound traffic, which enhances performance in download scenarios but not in upload scenarios, such as data backups and live streaming. Second, the LB remains on the data path, so special events on the LB, such as upgrades or failures, can potentially disrupt ongoing services. Lastly, different LBs may route flows from the same session to different servers, leading to inconsistencies in session state. LB in DSR still cannot prevent violations of PCC, which can cause data loss, performance issues, and a compromised user experience. Consequently, session synchronization remains a bottleneck for LB scalability.

For services that demand high performance, these bottlenecks become more pronounced. To achieve peak performance, it is crucial to bypass the LB from the data path of the connection. For example, Alibaba Cloud's RDS often requires bandwidth exceeding the terabyte level, while a single LB cluster has a secure bandwidth threshold of only 320 Gb. Using traditional ALB or ALB-DNAT modes would require multiple clusters to collectively provide service for RDS. To thoroughly address these issues and meet the needs of new products, we propose allowing traffic to bypass the LB, enabling a direct connection between clients and servers. This approach frees the network bandwidth and connection count from the constraints of the LB, improving network latency and stability.

C. Challenges of Bypassing LB

Compared to the FNAT forwarding mode, bypassing the LB to route traffic within a VPC network presents the following challenges.

Challenge 1: Direct communications between clients and servers bypassing LB. In the classical FNAT mode, communications between servers and the LB occur within the underlying network infrastructure, while in the VPC network client is deployed in an overlay network. Transition to the underlay network is facilitated through the LB, which serves as an interface for backend server access. Furthermore, the LB is responsible for central management of the entire traffic flow, session information retention across various flows, and application of user-specific address translations to orchestrate the data routing pathways.

Nevertheless, in scenarios where bypassing the LB is desired for direct client-server interaction, ensuring them on the same network layer is essential in communication. Within the constraints of the conventional FNAT mode, the migration of backend servers to the overlay network poses significant challenges.

Challenge 2: Flow conflict caused by multi-tenant access to cloud services. Currently, data centers construct VPC networks using various tunneling technologies [37]. For example, in a VXLAN network, tenants can manage their own subnet structure, IP address ranges, and allocation methods [38]. This can result in two clients having the same virtual IP address and simultaneously accessing the same server. Consequently, two SYN packets with the same quintuple can reach the same server. When these SYN packets undergo VXLAN decapsulation by the LB, they are restored to their original Ethernet frame format. Such packets with the same quintuple reaching the server will cause competition between the two clients and the server, allowing only one client to maintain a connection with the server at a time [39].

To address this issue, the current approach is to perform VXLAN decapsulation at the LB and use the LB's IP address as the source IP for NAT. However, using the LB's IP address as the source IP results in the loss of the client's IP address, making direct replies to the client impossible. Therefore, the key to achieving traffic bypass is to implement VXLAN decapsulation and Network Address Translation (NAT) without relying on the LB.

Challenge 3: Flow management in high-bandwidth applications. A server may need to maintain millions of concurrent connections, potentially resulting in tens or even hundreds of millions of connections over time. This can lead to many idle connections, as configuring TCP timeouts adaptively for various services is challenging. Improper timeout configurations can impact network performance [40]. Also, a million idle TCP connections could use approximately 4GB of memory, potentially affecting server connection speed and throughput [41].

To manage flows, current data centers use LBs to orchestrate bidirectional TCP connections, accommodating connection timeouts for various application flows. However, when bypassing the LB, clients connect directly with servers, lacking a centralized component to manage all flows. In high-traffic scenarios where NGLB is used, it is essential to release the resources of idle connections promptly.

D. Goals for NGLB

We believe that in order to accommodate the continuously increasing traffic and provide high-performance, low-latency services, it is necessary to design an LB with Full Bypassing. The NGLB must meet the following three requirements:

- The LB should be as removed from the data path as possible to reduce latency by direct packet transmission between the clients and the servers.
- The NGLB must be capable of managing a large number of tenants in virtualized networks while ensuring reliable connection quality.
- In high-bandwidth scenarios, the NGLB should manage millions of flows and promptly release idle connections to protect the availability of backend resources.

III. Design Overview

A detailed diagram of NGLB architecture is presented in Figure 3. The Alibaba Virtual Switch (AVS) is a customized

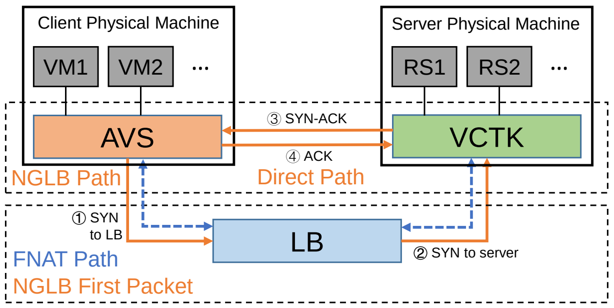


Fig. 3: The Overview of NGLB Architecture. The solid arrows illustrate the packet flow direction in NGLB, where only the initial SYN packet from the client to the server traverses the LB. The dashed arrows represent the FNAT data path, where the bidirectional traffic passes through LB.

Open vSwitch that manages the incoming and outgoing traffic of a node. In other words, the traffic from the Virtual Machines (VMs) deployed on the same physical node is managed by the AVS hosted on that node. Next, we implement a Virtual Connections Tracking Kit (VCTK) module in the kernel of the server's physical machine to manage communications with the Real Servers (RS) on that machine. AVS provides high flexibility and automatic configuration for managing VMs. Therefore, in response to dynamic user VMs, AVS has become the primary solution for managing VMs in data centers. We add new rules to facilitate the forwarding of NGLB traffic. In contrast, the deployment of RS is more stable and seldom requires dynamic adjustments. Meanwhile, VCTK, serving as a module for NGLB, minimizes its impact on servers by being mounted in the kernel.

Workflow. Connection information is transferred through TCP's three-way handshake process. As shown in Figure 3, the processing procedures are as follows: 1) The first packet path mirrors the FNAT mode: the client initiates a connection by sending a SYN packet to the LB. 2) The LB, following its load balancing strategy, selects a backend server and forwards the SYN packet. 3) The backend server, equipped with VCTK, decodes the SYN packet to extract client information and establishes a session. The server then sends a SYN-ACK packet back to the client. 4) AVS subsequently updates the flow table after extracting the server information. When the client's ACK packet passes through AVS, it is directly forwarded to the server, thereby completing the TCP three-way handshake and establishing a direct connection path between the client and server.

To facilitate direct communications (§IV) between clients and servers within the VPC network (§V), and also cater to the demands of high-bandwidth services (§VI), the following foundational design is proposed.

Active address learning. To enable direct communications between clients and servers within the VPC network, we have designed a proactive address learning mechanism in NGLB. This mechanism allows the AVS and VCTK to actively gather information from data packets and maintain a session table. By doing so, it equalizes the network plane for clients and servers, removing the LB from the data path. As a result, the LB only needs to perform load-balancing tasks and does not need to act as a bridge for address translation or maintain session information for each connection.

Multi-tenant isolation mechanism. To prevent TCP/IP address conflicts when multiple VPCs connect to the same server, we introduce a multi-tenant isolation mechanism in the server control plane. Servers manage a local IP table for NGLB flows, reallocating source addresses upon arrival to avoid conflicts. This mechanism allows users from different VPCs to access the same backend server, ensuring NGLB remains functional within VPC networks.

Hierarchical connections cleaner. In order to ensure the availability of backend resources when facing high-traffic services, we designed a method for managing NGLB traffic that does not rely on LB, named hierarchical connection cleaner. It operates within the server's kernel, tracking TCP connection data and setting fine-grained TCP timeout intervals to timely terminate idle TCP connections. This strategy considerably heightens server resource efficiency and fortifies the reliability of backend servers, particularly in high-bandwidth scenarios.

IV. Active Address Learning

To overcome the first challenge, we design forwarding modules for both clients and servers and a new LB forwarding mechanism for NGLB to ensure the direct data path. We use a layer-2 VXLAN tunnel and an active address learning mechanism to gather destination addresses and connectivity data, enabling direct client-server communication without affecting existing services. For traffic received by client and server virtual machines, the LB remains the intermediate communication point, ensuring full compatibility with diverse network architectures.

Address learning overview. We employ TCP's three-way handshake to enable traffic bypassing the LB. Initially, the client sends a SYN packet to initiate a new connection, with a half-connection record saved in the client's AVS. As the SYN packet passes through the LB, a Fast Bypass TCP Option (FBT) is inserted to convey information such as the client's IP. Upon reaching the server, the SYN packet enables the establishment of a complete session entry by parsing the FBT through VCTK. Following this, the server sends a SYN-ACK packet directly to the client. By parsing this packet, the client obtains the server's address and completes the half-connection record in the AVS. Finally, the client sends an ACK packet to the server, thereby completing the TCP three-way handshake and establishing a direct communication path between the client and server.

A. *First Packet Processing on LB*

The processing of the first packet is the most crucial step in establishing a connection with NGLB. We do not rely on additional packets to transmit information but instead achieve the goal of exchanging the addresses of the client and server by inserting TCP options in the first packet through the LB.

Figure 4 illustrates the four primary steps involved in the LB processing of the first packet. **Step 1:** The LB begins by decapsulating the VXLAN header to access the inner TCP information. **Step 2:** It queries the LB's pre-configured VIP table. To ensure compatibility with different forwarding modes like NGLB and FNAT, this table stores mode configurations for various VIP interfaces. **Step 3:** Insert the FBT. For NGLB traffic, the LB stops performing NAT at the internal packet layer. Instead, it inserts specific information into the FBT, which is used to forward details like the client's physical IP to the backend server. The FBT includes a 16-byte block containing essential information like the backend server's address and port, the physical machine's source address, and the VXLAN Network Identifier (VNI). This streamlined structure ensures that backend servers can access client data without the traditional NAT process. Introducing FBT serves two purposes. First, with connection redirection enabled, servers need to respond directly to client requests and identify which physical machine and VPC network the client belongs to. Thus, it's crucial to log the client's host IP and corresponding VXLAN tunnel identifier. Second, FBT records the server's IP and port number specified by the LB, as the packet's internal destination IP is virtual, representing a class of cloud services mapped to multiple backend servers. It doesn't point to a specific server, so translating the virtual IP and port to the actual server IP and port delivering the services is essential. **Step 4:** Encapsulate the VXLAN header. The LB encapsulates the data packet with a new VXLAN header, specifying the client's physical machine address as the source and the selected backend server's physical machine address as the destination. Finally, it dispatches the packet to the server's physical machine based on the predetermined load-balancing strategy, ensuring efficient routing while adhering to the load-balancing plan.

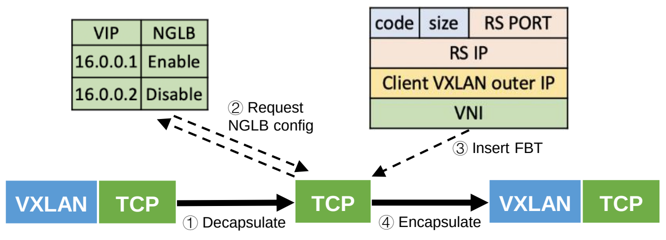


Fig. 4: First packet processing in LB

B. *Packets Redirection*

AVS enable learning server addresses from data packets. The AVS is implemented on the kernel of the physical machine and manages dozens of virtual machines, which are rented by different tenants. Figure 5 illustrates the operational workflow of the AVS module. When a client initiates a connection to the cloud service, AVS consults the routing table to determine whether it is a new connection. Then, AVS encapsulates the message within a VXLAN header and forwards it to the LB, triggering a new connection request. The LB processes this incoming traffic and forwards it to the designated server. Subsequently, the server, upon processing the traffic and identifying the client's address, returns the traffic directly to the client via the VXLAN tunnel, thus circumventing the LB.

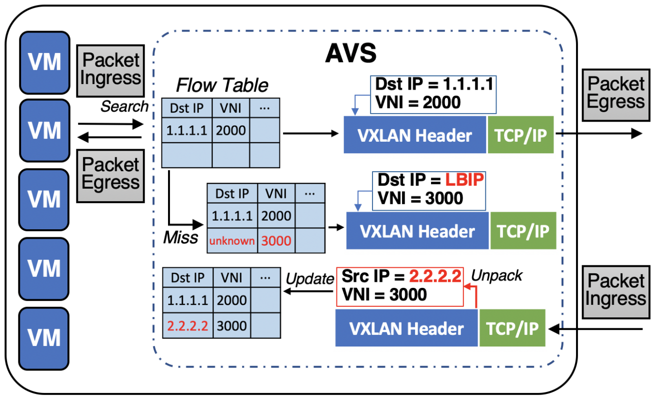


Fig. 5: Address learning and session table update in AVS

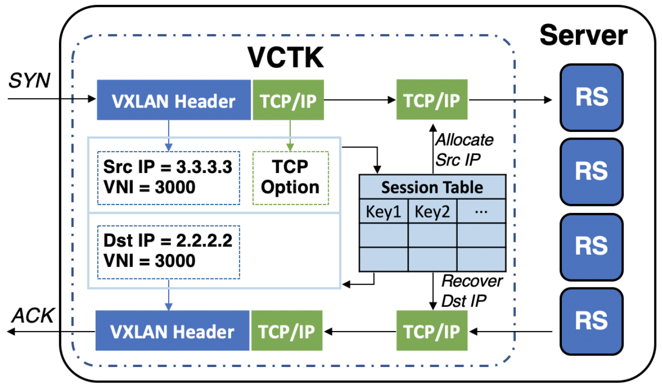


Fig. 6: Packets processing in VCTK

Since the client only has the address of the LB, it is unable to communicate directly with the server. For this, we implement a flow-based address resolution mechanism within AVS, enabling destination address learning. Upon receiving a response from the server, AVS matches it with the routing table to locate the corresponding half-connection record. Once a successful match is made, AVS strips away the VXLAN header to reveal the server's actual address and proceeds to update the session table by replacing the initial destination address (the LB's address) with the new address (the server's address). This active address learning process ensures that all subsequent connection traffic passing through AVS is encapsulated with the updated VXLAN destination address (the server's address), thus maintaining a bidirectional traffic flow independent of the LB.

VCTK module. The VCTK module is deployed on the server side in NGLB and achieves packet redirection through three core functions: processing of VXLAN header, processing of TCP options, and logging of session information. Figure 6 depicts the VCTK module's functionality within the server. The VCTK module first decapsulates the VXLAN header, recording the inner TCP source and destination addresses. It then analyzes the FBT options to obtain the client physical machine's VXLAN IP, VNI, and the RS address allocated by the LB. This information is logged into the session table to form a complete session entry. Finally, after executing the address translation, the packet is forwarded to the RS.

As the server response packets pass the VCTK module, they match with the established connection entries within the session table. Then, VCTK encapsulates the VXLAN header, which designates the server's physical machine address as the source and the client's physical machine address as the destination. Subsequently, the response is forwarded straight to the client via the VXLAN tunnel, effectively circumventing the LB and streamlining the communication process.

V. Multi-Tenant Isolation Mechanism

In this section, in order to solve the second challenge in §II-C, we detail the connection mechanism for multi-tenant service scenarios in data centers, called multi-tenant communication mechanism.

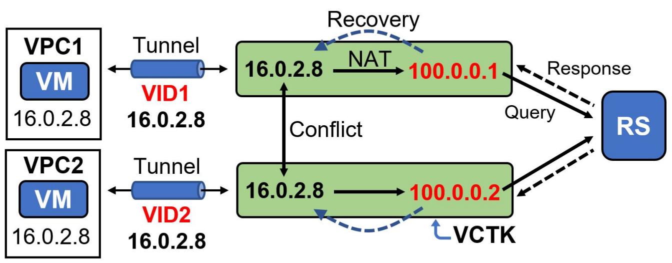


Fig. 7: Address mapping in isolation mechanism

The most straightforward solution to address multi-tenancy issues is to maintain a global flow table and reallocate addresses when flow conflicts occur. However, in a single cluster in Alibaba Cloud, the number of concurrent connections can reach up to 170 million, making the query latency significant. Therefore, we develop a distributed and lightweight layer-4 address isolation mechanism in NGLB.

A. *Isolation Mechanism*

In FNAT, packets leave the VXLAN tunnel at the LB, and NAT is performed here to prevent flow conflicts. Therefore, we have integrated this mechanism into VCTK, where packets leavethe VXLAN tunnel, ensuring that all flows accessing the current server do not conflict by isolating them before the packet completely loses its VNI information.

When the flow enters the server, VCTK records the flow information, which is detailed in §IV-B. Before forwarding to RS, VCTK first performs an address translation on the packet, changing the destination address to the RS address extracted from FBT. As shown in Figure 7, due to the absence of VNI information, 5-tuple conflicts may occur at this time. In the figure, we only show the scenario of source address conflicts. However, actual conflicts occur when the entire 5-tuple in the TCP packet is identical. Therefore, we perform an additional address translation to eliminate flow conflicts. In VCTK, we assign a local address to each flow, the scope of which is only for the current physical machine. This address is maintained in the VCTK session table and is associated with the flow's 5-tuple and VNI, ensuring non-conflicting allocation. Theoretically, an individual VCTK is capable of holding up to 4 million sessions, but in the actual environment

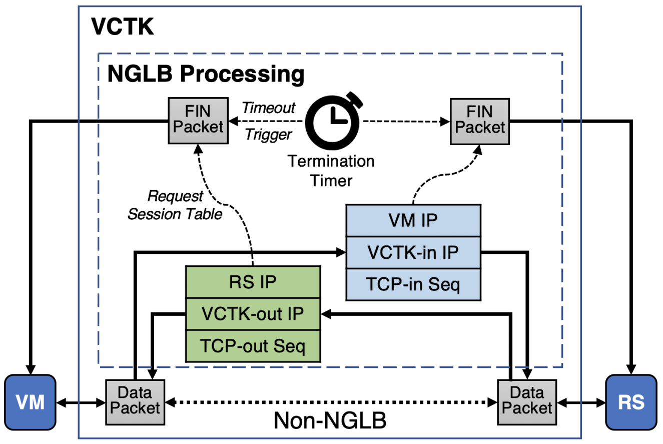


Fig. 8: Active termination mechanism on VCTK

of the Alibaba data center, the average per physical machine is approximately 0.3 million sessions. Each session entry is composed of two flow pointers along with the pertinent session data. Employing a hash table for storage achieves microsecond-level query responses, which satisfy the demand for low latency.

By an additional source address translation, converting the source address to the locally assigned address, the issue of flow conflicts is solved. Later, when RS completes processing and answers the request, the data packet passes through the VCTK module again, hitting the corresponding session entry, and restores the locally assigned address to the original VM address, thus achieving isolation of multi-tenant traffic without effect on the original communication.

VI. Hierarchical Connection Cleaner

In this section, we address the third challenge outlined in §II-C. We present a flow management scheme designed for NGLB, called the hierarchical connection cleaner, which enables us to manage flows for NGLB using VCTK instead of the LB.

As the LB functionality is offloaded to VCTK, the timeout termination of connections on the LB becomes invalid. Unlike LB, VCTK is a lightweight module that lacks the capability to maintain TCP connections, as it will occupy significant memory. Additionally, the backend servers are designed to handle a diverse range of traffic, servicing not only NGLB traffic. A simplistic adjustment of the TCP timeout settings within RS would result in premature termination of all traffic, undermining the universality of RS.

Therefore, we design a traffic management mechanism in VCTK, capable of specifically managing the traffic of NGLB without modifying global TCP parameters, such as TCP timeout. This method offers refined control and optimization of traffic, ensuring that NGLB can efficiently maintain its stability amidst the constant rise in network traffic.

A. Active Termination Mechanism

We achieve the management of connections in VCTK by recording partial information of TCP connections, including the VM addresses, RS addresses, and the bidirectional TCP sequence numbers etc. Consequently, VCTK has the capability to communicate separately with both VM and RS without the necessity of establishing actual TCP connections. This design grants VCTK a level of control over the traffic, ensuring efficient resource utilization without significant consumption.

Traffic hierarchy. We propose a traffic hierarchy strategy that can apply different control schemes to different traffic. Figure 8 demonstrates VCTK's hierarchical strategy towards managing distinct traffic types. (1) Non-NGLB traffic: The LB controls the traffic, so VCTK does not need to interfere with this traffic. It simply forwards it directly to RS for processing. (2) NGLB traffic: VCTK records key information such as the TCP source, destination addresses, and the sequence numbers. Based on predetermined settings, it assigns varying timeout durations for flows serving different services. When a flow remains idle beyond its timeout threshold, the active connection termination will be activated to disconnect the connection.

Active connection termination. We designate the communication direction from client VM to server RS as incoming. Illustrated in Figure 8, when a flow's idle period surpasses a predetermined threshold, it triggers the generation of termination packets. To guarantee that both connection ends can properly receive these packets, VCTK generates two legitimate packets using the 5-tuples and sequence numbers logged in the session table. These packets are of the TCP RST type. VCTK sends these packets to both the client VM and the server's RS. Upon their reception of the RST packet, the connection is swiftly terminated, thereby freeing up the associated resources.

Therefore, in VCTK, we have greater flexibility in setting the connection keep-alive times for different flows. This means we can assign appropriate timeout periods for each TCP connection based on different service needs. As a result, when a connection becomes inactive, it can promptly release the resources it occupies. This not only improves the utilization of server resources but also optimizes the overall network performance, ensuring that our services remain efficient and stable even when faced with a large number of concurrent connections.

B. Running example

NGLB was implemented in a distributed manner, where the VCTK module was implemented in the kernel with ∼200 lines of C code, AVS with ∼100 lines of C code, and LB with ∼100 lines of C code. To illustrate the design of NGLB, we use the following running example.

Running example. Figure 9 illustrates operations on the packets as they pass through three modules during the TCP three-way handshake in NGLB. First, the client (CIP) initiates a service request to the LB (VIP). AVS receives the SYN packet and adds the "route flag" in the VXLAN header. The outer source IP is set as the client NIC address (HIP), and the outer destination IP is set as the LB cluster NIC address (LIP) (step b). After receiving SYN packets, the LB cluster allocates them to specific LBs based on the VIP carried by the packets. LB leverages the conditions of the back-end servers while making the scheduling decisions to a server (RIP) and

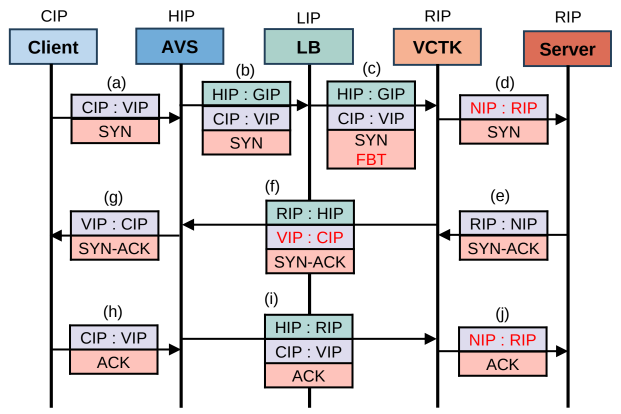
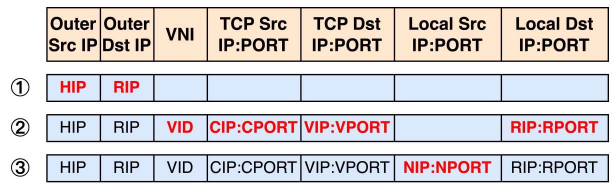


Fig. 9: In the Full Bypassing scheme, the packet transformation of TCP handshake with NGLB. Only steps (b) and (c) require passing through the LB. Steps (f) and (i) bypass the LB entirely and communicate directly between clients and servers.



	Outer Src IP	Outer Dst IP	VNI	TCP Src IP:PORT	TCP Dst IP:PORT	Local Src IP:PORT	Local Dst IP:PORT
①	HIP	RIP					
②	HIP	RIP	VID	CIP:CPORT	VIP:VPORT		RIP:RPORT
③	HIP	RIP	VID	CIP:CPORT	VIP:VPORT	NIP:NPORT	RIP:RPORT

Fig. 10: The session table in VCTK

recognizes the "route flag" and adds FBT in the TCP header (detailed in the description of LB §IV-A). Finally, LB sets the outer source IP and the outer destination IP of the VXLAN header to LIP and RIP (step c).

Upon receiving the SYN packet, VCTK creates a new session in the tables (see figure 10). Then, a series of operations is performed to fill each field. Firstly, it takes out HIP, RIP in the VXLAN header, and inserts these three pieces of information into the Outer Src IP, Outer Dst IP, and VNI field (See Figure 10 step ①). Secondly, VCTK extracts CIP:CPORT, VIP:VPORT from the inner TCP header of SYN packets, as well as HIP, RIP:RPORT, and client tunnel identifier VID from FBT, and inserts them into the session table (See Figure 10 step ②). Thirdly, utilizing VID and CIP:CPORT from the session table, VCTK allocates a unique NIP:NPORT for this connection and inserts it into the session table (See Figure 10 step ③). Finally, based on the session table, VCTK translates the source and destination IP:PORT from CIP:CPORT and VIP:VPORT to NIP:NPORT and RIP:RPORT within the inner data frames of the SYN packet (step d).

The server responds with a SYN+ACK packet, including the packet's source address (RIP and RPORT) and destination address (NIP and NPORT) respectively (step e). Then VCTK converts the inner source and destination address of the SYN+ACK packet from RIP:RPORT and NIP:NPORT to VIP:VPORT and CIP:CPORT based on the session table. It then encapsulates the packet with VXLAN and sets the outer source IP and destination IP as RIP and HIP. Finally, the packet can be directly sent to the client host (step f).

AVS, upon receiving the SYN+ACK packet, decapsulates the VXLAN header, updates the outer destination address of the connection in the session table from LIP to RIP, and forwards the packet to the client (step g). The client replies with an ACK packet (step h). AVS forwards the ACK packet to the server while setting the outer destination IP and source IP as RIP and HIP (step i). VCTK matches the connection, decapsulates the VXLAN header, executes NAT from CIP:CPORT and VIP:VPORT to NIP:NPORT and RIP:RPORT, and sends the ACK packet to the server (step j). The subsequent transmission of data packets follows the same steps from e to j.

VII. Evaluation

In this section, we use two types of setups to evaluate the performance of NGLB. The first, *Large Scale Benchmarking*, examines the request latency of NGLB compared to another LB (i.e., in FNAT mode), both of which provide L4 load balancing services for the applications in Alibaba Cloud. Next, we create an isolated environment to benchmark the performance of each component of NGLB, including throughput and resource utilization, such as CPU overhead.

A. Large scale benchmarking

We first present the evaluation of the OSS service currently running on Alibaba Cloud. Figure 11 illustrates the request latency for uploading (PUT) and downloading (GET) 100 KB files via Alibaba Cloud's Object Storage Service (OSS). In the production environment, NGLB reduces average upload latency by 18% and 12% compared to FNAT and DSR, respectively. For downloads, it achieves a 10% reduction in average latency.

We collected data on NGLB and FNAT from a cluster consisting of 475 physical machines with 26 cores each. The analysis reveals that NGLB supports a substantial 8 Tbps of data traffic using only 100,000 listeners, which monitor and forward client traffic to the backend servers. In contrast, FNAT required 300,000 listeners to handle 10 Tbps of traffic. While these results include additional bandwidth set aside for redundancy to ensure service reliability, it is clear that NGLB is more efficient in its utilization of load-balancing resources when handling equivalent amounts of bandwidth.

B. Mirobenchmark experiments

Experimental setup. To thoroughly evaluate the performance of each component, we establish an isolated network environment that separates our test bed from other cloud services. The test bed consists of 10 VMs, each equipped with a 2.50GHz CPU and a single core, to host the clients. The cloud services are deployed on a physical machine featuring an Xeon E5-2630 CPU at 2.30GHz with 10 cores. The LBs are installed on another physical machine with an Xeon E5-2630 CPU at 2.30GHz and 12 cores; however, we restrict the LB operation to just one core to test the upper performance limits of the LB. In the performance of NGLB, we consider the latency, throughput, and CPU overhead. To better saturate the system, we do not possess a full TCP client. Alternatively, we build a DPDK-based client that sends SYN and ACK packets to stress test the LB and server.

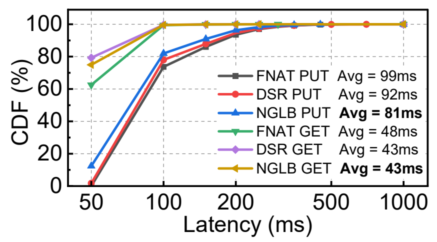


Fig. 11: The uploading and downloading latency of online OSS service

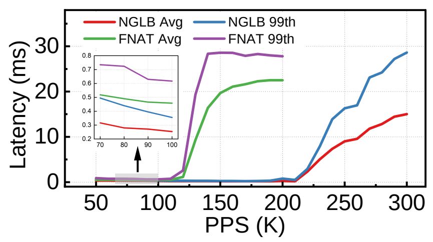


Fig. 12: Reduction in RTT through bypassing LB

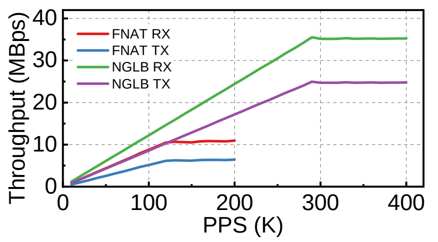
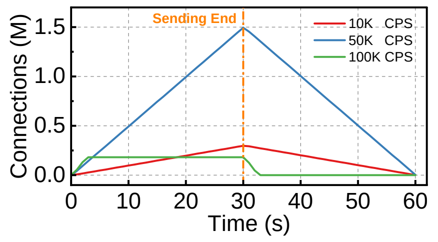
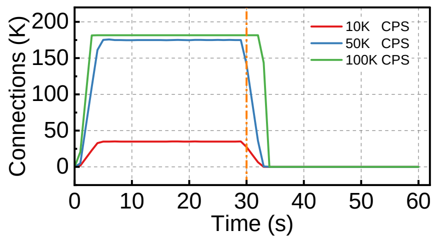


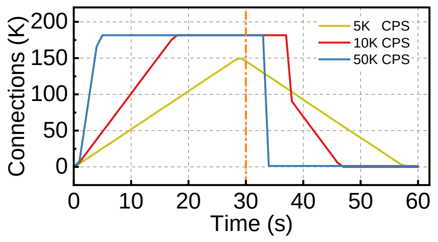
Fig. 13: Throughput improvement through bypassing LB



(a) NGLB connections in VCTK



(b) NGLB connections in LB



(c) Baseline connections in LB

Fig. 14: Experiments measure the connections maintained by LB and VCTK. Figure (a) shows the number of NGLB connections maintained by VCTK under different CPS. Figure (b) shows the number of NGLB connections recorded by LB under different CPS. Figure (c) shows the number of FNAT connections maintained by LB under different CPS.

1) *Effectiveness of NGLB:*

RTT. Figure 12 shows the RTT of NGLB when packets request backend services under different configurations of PPS. We assess latency by measuring the average RTT and the 99th-percentile RTT. NGLB substantially reduces RTT by circumventing LB. Within the threshold of 100K packets per second (PPS), both NGLB and FNAT maintain low-latency services, yet NGLB achieves a latency reduction of around 40%. Furthermore, NGLB doubles the number of packets handled per second by eliminating traffic bottlenecks caused by the LB. Due to the LB bottleneck, the RTT increases at 110K PPS for FNAT. In contrast, NGLB maintains low latency until it reaches 210K PPS, at which point latency increases due to backend server saturation.

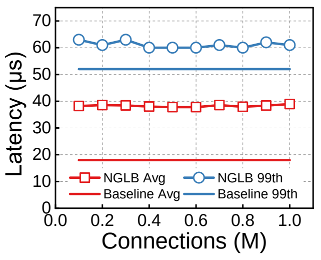
Throughput. Figure 13 illustrates the throughput of FNAT and NGLB under various PPS configurations. We conduct this test with 128-byte packets, collecting data on both the receive (RX) and transmit (TX) throughput on the server's Network Interface Card (NIC). Here, receiving refers to the direction from the client to the server, and transmitting refers to the direction from the server to the client. The throughput bottleneck occurs at 110K PPS with a total throughput of 17 Mbps. However, NGLB encounters the throughput bottleneck under a higher load of 290K PPS, and the total throughput reaches up to 60 Mbps, bringing nearly a 3$\times$ performance improvement. Furthermore, the throughput bottleneck of NGLB exceeds the point where RTT starts to increase, which means that even at 210K PPS when RTT begins to rise, the system has not yet reached the backend server's loading limit. This phenomenon indicates that NGLB leaves extra bandwidth redundancy for the server, allowing it to effectively handle abnormal fluctuations or surges in network traffic, enhancing network resilience and stability.

2) *Effectiveness of handling TCP connection:*

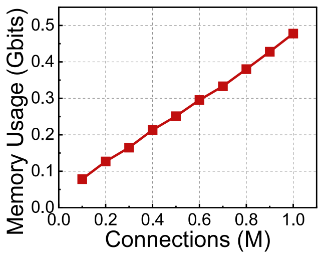
Our objective is to assess the effect of deploying NGLB on the establishment and maintenance of TCP connections. We initiate a large volume of TCP connections by generating numerous SYN packets. The test lasts for 60 seconds, which is sufficient to bring both the LB and the server to a state of full load. We stop establishing new connections at the 30th second and continue to collect data for an additional 30 seconds to record the process of connection resource release.

Created connections per second. NGLB significantly reduces the complexity of LB and enhances the capacity of LB to establish new connections. Figure 14 shows the number of TCP connections that NGLB and FNAT can maintain, reflecting the system's capacity in session maintenance. The data reveals that NGLB can maintain a stable high-speed rate of 50K new connections per second, however, FNAT can only maintain 5K per second.

For FNAT, the CPS are configured to 5K, 10K, and 50K, while for NGLB, the settings are 10K, 50K, and 100K, respectively. The LB of NGLB is able to timely release completed connection resources to provide space for new connection requests, a feature attributed to the mechanism in NGLB design, where LB does not need to maintain connection states. However, the LB of FNAT needs to maintain every connection, and when the connection resources of LB are exhausted, it can not continue to establish new connections. To test the performance of the LB of FNAT, we set the connections per second (CPS) to 5K, 10K, and 50K, respectively. For NGLB, the CPS is set to 10K, 50K, and 100K. As shown in Figure 14c, the LB of FNAT can only handle 5K CPS. When the CPS increases to 10K, LB hits the bottleneck. In contrast,



(a) Server process latency



(b) VCTK memory usage

Fig. 15: Experiments measure the VCTK resource consumption by process latency and memory usage. Figure (a) shows the latency while the server handles NGLB traffic through VCTK under different numbers of connections. Figure (b) shows the memory usage of VCTK while maintaining different numbers of connections.

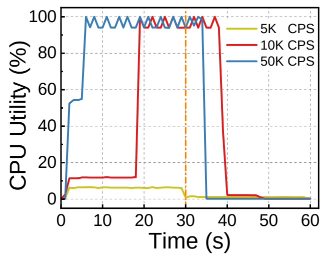
under the configuration of 50K CPS, the LB of NGLB can handle all connection requests, with the ability to process new connections over 10× higher than the LB of FNAT. However, when CPS increased to 100K, as shown in Figure 14a, the LB reached the bottleneck in just 4 seconds. This is because a large number of connection requests blocked the LB in a short period of time, making it impossible for the LB to timely release the expired connections and accept new ones. Eventually, the LB only successfully established 180K connections, far fewer than the other two configurations.

3) The overheads of each component:

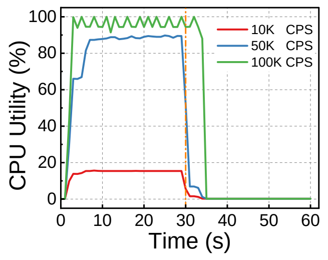
We use server processing latency, memory usage, and CPU usage to evaluate the resource overhead of the VCTK module and LB.

The latency of VCTK. We test the processing latency of servers deployed with VCTK by measuring the time taken for data packets to traverse from entry to exit through the server's NIC. Figure 15a shows the processing latency of servers while VCTK sustains different numbers of connections, with the dashed line representing the latency of servers not deployed with VCTK, providing a baseline for comparison. The result illustrates that the processing time does not increase substantially as the number of VCTK-maintained connections grows, demonstrating that an abundance of connections does not markedly deteriorate performance. Additionally, NGLB exhibits a modest latency increase of 10 to 20 microseconds over the baseline, which is justified by the need for additional handling of data packets. Nonetheless, these added time expenses pale in comparison to the decrease in RTT, rendering them insignificant.

The memory usage of VCTK. Figure 15b shows the memory usage of VCTK while maintaining different numbers of connections. According to the design of VCTK's session table, each connection occupies about 60 bytes of space to store session information, which is consistent with the experimental results. The memory usage of VCTK increases linearly with the number of connections, with the data indicating that 1 million session entries occupy 490 Mbits of memory.



(a) LB CPU usage in FNAT



(b) LB CPU usage in NGLB

Fig. 16: LB CPU usage. Figure (a) shows the CPU usage of LB under different CPS in NGLB. Figure (b) shows the CPU usage of LB under different CPS in FNAT.

The resource consumption of LB. We evaluate the performance of the LB by conducting CPS testing and measuring its efficiency through the monitoring of the LB's CPU utilization under various CPS configurations. A sharp increase to 100% CPU usage indicates significant stress on the LB, signifying its incapacity to handle additional requests.

NGLB significantly reduces the CPU utilization of LB and greatly improves the resource utilization rate of LB. Comparison between Figure 16a and 16b, under the same 10K CPS configuration, the LB based on FNAT is overloaded, while the LB based on NGLB only uses 15% of the CPU to handle all connection requests. When the connection requests increase to 50K CPS, the LB of FNAT becomes overloaded within just 4 seconds, whereas NGLB operates stably with about 88% CPU utilization.

The results indicate that the LB of FNAT can only handle 5K CPS; although the CPU utilization seems low, once the connection resources of the LB are fully occupied, it will lead to a sharp increase in CPU utilization, resulting in a significant waste of CPU computational resources. In contrast, the LB of NGLB is able to fully utilize the CPU for connection processing, which enables it to handle connections up to 10× higher than that of the LB with FNAT architecture.

VIII. Experiences

NGLB has been implemented across our extensive VPC for several years, significantly enhancing the robustness of Alibaba's cloud network. Despite challenges such as growing network sizes, fluctuating traffic, and cyber threats, NGLB consistently delivers superior performance to our users. We are confident in our strategic initiatives. In addition to the strategies outlined in this document, our extensive experience in research and development for cloud networks has greatly contributed to our expertise, which we are eager to share in this section.

A. The benefit of using distributed load balancing in cloud data centers

Improved Resource Management. Distributed design is set to play a pivotal role in enhancing the performance of cloud services. As the pace of hardware performance gains decelerates, it is becoming increasingly difficult to improve the service quality through optimization of isolated components. Take

NGLB as an instance, without offloading traffic, an individual heavy-traffic service would occupy multiple LB instances. By the coordination between VCTK and AVS components, although the approach does not reduce the resources needed for traffic maintenance, it leverages a distributed control strategy to significantly minimize the complexity of maintaining LB clusters.

Rapid Service Deployment. The modularized deployment significantly enhances the system's flexibility, while the integrated approach to functionalities minimizes inter-module coupling. In traditional data centers, integrating all packet header processing in the LB necessitates session synchronization functionality, one of the reasons why data center LBs cannot be infinitely scaled. In NGLB, by decoupling the LB and integrating similar functionalities on the same side, the restrictions on LB scaling are lifted, while also laying the groundwork for rapid module iteration.

B. Issues caused by state sinking

Offloading the data path from the LB to the client and server makes the LB unaware of the state of the connections. This will give rise to the following issues for discussion, and can serve as directions for future research.

Can NGLB deploy outside Alibaba Cloud? To promote broader adoption beyond Alibaba Cloud, NGLB is designed to be platform-agnostic and free from dependencies on proprietary hardware or software. This design choice ensures compatibility with a wide range of cloud environments. In typical data center settings, each physical client host runs multiple VMs, which are uniformly managed through Open Virtual Switch (OVS). The client-side active address learning algorithm is built on top of standard OVS functionality and does not rely on specialized hardware. On the LB side, packet encapsulation is performed using TCP options at the protocol stack level, maintaining full compliance with standard networking protocols. On the server side, the VCTK is entirely implemented within the Linux kernel's packet processing pipeline. This enables efficient in-kernel execution while avoiding the need for dedicated hardware or modifications beyond standard Linux interfaces. Consequently, all components of NGLB are inherently portable and can be deployed in diverse data center environments using commodity infrastructure.

This architecture provides two primary advantages: (1) high adaptability to various cloud architectures, and (2) simplified deployment across heterogeneous infrastructures without requiring hardware changes to existing platforms.

How does NGLB ensure safety? A primary risk of direct client-to-server connections is the exposure of server IP addresses, potentially enabling targeted attacks. In NGLB, this risk is mitigated by the client-side AVS, a component fully managed by the cloud provider. Server addresses are only accessible within AVS and remain hidden from user-space processes and end users, effectively preventing address leakage. Another concern is vulnerability to connection-based DDoS attacks, such as SYN floods. To address this, NGLB integrates a lightweight SYN-proxy mechanism at the LB, which monitors SYN packets and applies rate limiting. The LB also collaborates with backend servers to detect TCP backlog saturation. When saturation occurs, the system activates SYN-proxy mode, temporarily reverting to a centralized LB path to absorb and filter malicious traffic. While effective against SYN floods, this mechanism does not address application-layer or large-scale volumetric attacks, which require complementary solutions such as WAFs or application-level rate limiting. This limitation, compared to centralized LB designs, is acknowledged and targeted for future improvement.

How does NGLB handle failures? NGLB incorporates mechanisms to maintain service continuity in the presence of component failures. When a VCTK or server instance fails, the LB detects the issue via periodic health checks and stops routing new connections to the affected node. Established connections rely on standard TCP timeouts for recovery, or a recovered VCTK can send TCP RST packets to reconnect. While such failures are rare, they are expected at scale and are handled gracefully. In the case of AVS failure, packet forwarding is disrupted, and traffic automatically falls back to the load balancer in FNAT mode, preserving session continuity. These mechanisms together enhance the fault tolerance of the system while retaining the performance advantages of direct connection handling.

Can the design of NGLB collect the network statistics in real time? Currently, as NGLB offloads traffic and bypasses the traditional LB, the network monitoring functions on the LB are no longer applicable. We use its extensive bandwidth to support high-traffic services with NGLB, eliminating the need for traditional network monitoring to prevent congestion. However, as services grow, network monitoring will still be needed for NGLB. Most LB functions have been moved to the server-side VCTK module in NGLB. In future work, we can add a network monitoring module to VCTK. Moreover, Alibaba Cloud already possesses a well-established network monitoring system and abundant experience, which will greatly assist in the deployment of distributed network monitoring on VCTK.

Can the design of NGLB support live migration? Live migration is a common phenomenon in cloud services, which increases flexibility [42]. The VM does not rely on a single physical server and allows for the migration of VMs to available physical servers before upgrades or other changes to the physical machine environment occur. In FNAT and DSR, clients are only required to send packets to the LB, which can effortlessly find the corresponding backend servers and return the response to the right clients. However, with NGLB, when a virtual machine migrates to a new physical machine, a lack of shared information between the new client's physical machine and the server's physical machine interrupts direct communication. To enable a live migration, we can simultaneously migrate the flow tables pertinent to the virtual machine to the AVS on the new physical machine. This method preserves the status of all active flows, ensuring connections are re-established after the migration.

IX. Related Work

At present, many researchers have reduced the load of LB by changing the transmission path of data packets, thereby reducing service latency and avoiding LB from becoming bottleneck to improve data center performance.

Direct Server Return. Ananta [5] uses Multiplexer (Mux) and Host Agent (HA) to implement DSR. It sets up a VIP in Mux and deploys a load balancing algorithm for scheduling, while the HA on the server side is responsible for NAT and redirection to achieve Mux is responsible for forwarding all inbound packets, and all outbound traffic bypasses Mux. Maglev implements DSR through Mux and the server. Compared with Ananta, Maglev's Mux [1] adds General Routing Encapsulation (GRE), while the server performs decapsulation and response redirection. Although the LB in DSR mode is involved only in the forwarding of inbound packets, if the LB fails, it still affects established TCP connections, which is a structural flaw in the DSR architecture. NGLB significantly reduces the impact of LB on connections by bidirectional traffic bypassing the LB.

Bypassing. Another structure was also first proposed by Ananta [5], which is that Mux only participates in the establishment of TCP connections, and subsequent TCP communication will completely bypass Mux. However, Fastpath in Ananta still requires Mux to participate in the complete TCP connection establishment. CHEETAH [16], [22] bypasses the state matching in the LB by inserting a cookie encoded with the server ID into the packets. However, the packets of CHEETAH still require forwarding through the LB.

The most relevant works to us are as follows. R2P2 [43] exposes the RPC abstraction to the endpoints and the network, enabling servers to bypass routers and directly respond to the RPC. It achieves load balancing in the router by allowing servers to proactively report their own load conditions. The work most closely aligned with our research is CRAB [35], where the authors propose leveraging TCP options to convey client information, thereby facilitating traffic bypass of load balancers. Nevertheless, CRAB is limited to local area network (LAN) communications, rendering it impractical for deployment in large-scale network environments. To address this limitation, we integrate VXLAN technology through coordinated module interactions, such as VCTK. This integration enables NGLB to achieve seamless deployment in expansive, data center-scale networks.

X. Conclusion

As the demand for cloud computing continues to grow, data center interfaces must handle millions of new connections and nearly 1TB of traffic per second. This immense load challenges the processing speed and stability of these interfaces. To address this, we propose the NGLB architecture, which offloads a significant amount of traffic to the Alibaba Cloud load balancing interface. This approach significantly reduces the complexity of the LB and lowers the latency for users accessing cloud services.

In this paper, we introduce the establishment of direct communication between clients and servers through active address learning. We also proposed a multi-tenant isolation scheme to resolve conflicts arising from multiple tenants accessing cloud services in data centers, making the architecture suitable for deployment in cloud environments. To ensure NGLB meets the demands of high-bandwidth applications, we designed a lightweight connection cleanup mechanism in VCTK that quickly releases backend service resources. Finally, we shared our experience deploying NGLB, hoping it will benefit other cloud providers facing similar challenges. This work demonstrates the potential of NGLB to enhance the efficiency and reliability of cloud data center operations.

References

[1] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.

[2] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.

[3] R. Gandhi, Y. C. Hu, C.-k. Koh, H. H. Liu, and M. Zhang, "Rubik: Unlocking the power of locality and end-point flexibility in cloud scale load balancing," in *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, 2015, pp. 473–485.

[4] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu, "Stateless datacenter load-balancing with beamer," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 125–139.

[5] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu *et al.*, "Ananta: Cloud scale load balancing," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 207–218, 2013.

[6] S. Shi, Y. Yu, M. Xie, X. Li, X. Li, Y. Zhang, and C. Qian, "Concury: A fast and light-weight software cloud load balancer," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 179–192.

[7] R. Cohen, M. Kadosh, A. Lo, and Q. Sayah, "Lb scalability: Achieving the right balance between being stateful and stateless," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 382–393, 2021.

[8] A. Aghdai, C.-Y. Chu, Y. Xu, D. H. Dai, J. Xu, and H. J. Chao, "Spotlight: Scalable transport layer load balancing for data center networks," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 2131–2145, 2020.

[9] G. Soni and M. Kalra, "A novel approach for load balancing in cloud data center," in *2014 IEEE international advance computing conference (IACC)*. IEEE, 2014, pp. 807–812.

[10] S.-L. Chen, Y.-Y. Chen, and S.-H. Kuo, "Clb: A novel load balancing architecture and algorithm for cloud services," *Computers & Electrical Engineering*, vol. 58, pp. 154–160, 2017.

[11] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, 2017.

[12] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 15–28.

[13] J. Zhang, S. Wen, J. Zhang, H. Chai, T. Pan, T. Huang, L. Zhang, Y. Liu, and F. R. Yu, "Fast switch-based load balancer considering application server states," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1391–1404, 2020.

[14] T. Muhammad, "A comprehensive study on software-defined load balancers: Architectural flexibility & application service delivery in on-premises ecosystems," *International Journal of Computer Science and Technology*, vol. 6, no. 1, pp. 1–24, 2022.

[15] M. Rahman, S. Iqbal, and J. Gao, "Load balancer as a service in cloud computing," in *2014 IEEE 8th international symposium on service oriented system engineering*. IEEE, 2014, pp. 204–211.

[16] T. Barbette, C. Tang, H. Yao, D. Kostic, G. Q. Maguire Jr, P. Papadimitratos, and M. Chiesa, "A high-speed load-balancer design with guaranteed per-connection-consistency." in *NSDI*, 2020, pp. 667–683.

[17] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-Aware cluster scheduling policies for deep learning workloads," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 481–498. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

[18] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476223

[19] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond {GPUs} for {DNN} scheduling on {Multi-Tenant} clusters," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 579–596.

[20] Q. Hu, M. Zhang, P. Sun, Y. Wen, and T. Zhang, "Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 457–472.

[21] A. F. S. Devaraj, M. Elhoseny, S. Dhanasekaran, E. L. Lydia, and K. Shankar, "Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments," *Journal of Parallel and Distributed Computing*, vol. 142, pp. 36–45, 2020.

[22] T. Barbette, E. Wu, D. Kostić, G. Q. Maguire, P. Papadimitratos, and M. Chiesa, "Cheetah: A high-speed programmable load-balancer framework with guaranteed per-connection-consistency," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 354–367, 2021.

[23] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, p. 68–73, Dec. 2009. [Online]. Available: https://doi.org/10.1145/1496091.1496103

[24] T. Ghosh, D. Sarkar, T. Sharma, A. Desai, and R. Bali, "Real time failure prediction of load balancers and firewalls," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2016, pp. 822–827.

[25] A. Bremler-Barr, E. Brosh, and M. Sides, "Ddos attack on cloud auto-scaling mechanisms," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[26] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 500–507.

[27] S. Taherizadeh and V. Stankovski, "Dynamic multi-level auto-scaling rules for containerized applications," *The Computer Journal*, vol. 62, no. 2, pp. 174–197, 2019.

[28] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand *et al.*, "Autopilot: workload autoscaling at google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[29] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Rfc 7348: virtual extensible local area network (vxlan): a framework for overlaying virtualized layer 2 networks over layer 3 networks," 2014.

[30] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on software engineering*, vol. 14, no. 2, pp. 141–154, 1988.

[31] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, 2008, pp. 51–62.

[32] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "Opennetvm: A platform for high performance network service chains," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, 2016, pp. 26–31.

[33] IBM, "Kubernetes networking: Load balancer and network load balancer." https://ibm.github.io/kubernetes-networking/services/loadbalancer/.

[34] Cilium, "Kubernetes without kube-proxy." https://docs.cilium.io/en/stable/network/kubernetes/kubeproxy-free/.

[35] M. Kogias, R. Iyer, and E. Bugnion, "Bypassing the load balancer without regrets," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 193–207.

[36] R. Yang and M. Kogias, "Heels: A host-enabled ebpf-based load balancing scheme," in *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*, 2023, pp. 77–83.

[37] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Transactions on parallel and distributed systems*, vol. 28, no. 1, pp. 290–304, 2016.

[38] K. Jeong, R. Figueiredo, and K. Ichikawa, "Pares: Packet rewriting on sdn-enabled edge switches for network virtualization in multi-tenant cloud data centers," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017, pp. 9–17.

[39] I. S. I. U. of Southern California, "Transmission control protocol," https://www.rfc-editor.org/rfc/rfc793.txt.

[40] T. Faber, J. Touch, and W. Yue, "The time-wait state in tcp and its effect on busy servers," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 3. IEEE, 1999, pp. 1573–1583.

[41] L. YU, "Improving quality of experience and protocol performance using user context information," Ph.D. dissertation, NATIONAL UNIVERSITY OF SINGAPORE, 2012.

[42] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, 2005, pp. 273–286.

[43] M. Kogias, G. Prekas, A. Ghosn, J. Fietz, and E. Bugnion, "R2p2: Making rpcs first-class datacenter citizens," in *Proceedings of the 2019 USENIX Annual Technical Conference*, no. CONF. USENIX ASSOC, 2019, pp. 863–879.

Yuchen Zhang received B.Eng from Zhejiang University in 2021. He is currently pursuing a Ph.D. in the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. His research interests include cloud networking and load balancing.

Shuai Jin is an Alibaba Cloud Technical Manager. His research interests include network virtualization, high-performance networking, and programmable networking.

Zhenyu Wen (Senior Member, IEEE) is currently the Tenure-Tracked Professor with the Institute of Cyberspace Security and College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. His research interests include the IoT, crowd sources, AI systems, and cloud computing.

Shibo He (Senior Member, IEEE) is a professor with the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. He also serves as the director of the Collaborative Innovation Center for Industrial Cyber-Physics Systems, which is jointly established by the provincial and ministerial authorities, and as the deputy director of the Industrial Control Research Institute. His research focuses on the Internet of Things (IoT) and Big Data analytics.

Qingzheng Hou is an Alibaba Cloud Senior Software Engineer. His research interests include network virtualization, high-performance networking, and programmable networking.

Yang Song is an Alibaba Cloud Network Architect. Her research interests include network virtualization, high-performance networking, and programmable networking.

Zhigang Zong is the director of VPC product development at Alibaba Cloud Network and has nearly 20 years of experience in data communication network research and development. Leading the 2.0 architecture upgrade of Alibaba Cloud Network Luoshen Platform. Through software and hardware integration and cloud native technology, it has changed the traditional cloud network SDN+NFV architecture and continues to lead in cloud network performance, scale, and elasticity.

Xiaomin Wu is an Alibaba Cloud Network Architect. His research interests include network virtualization, high-performance networking, and programmable networking.

Bengbeng Xue is an Alibaba Cloud Technical Manager. His research interests include network virtualization, high-performance networking, and programmable networking.

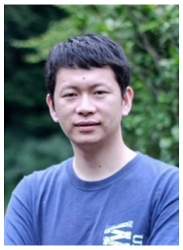
Chenghao Sun is an Alibaba Cloud Director. His research interests include network virtualization, high-performance networking, and programmable networking.

Ku Li is an Alibaba Cloud Director. His research interests include network virtualization, high-performance networking, and programmable networking.

Xing Li is an Alibaba Cloud Director. His research interests include network virtualization, high-performance networking, and programmable networking.

Biao Lyu is currently pursuing the Ph.D. degree in polytechnic with Zhejiang University, Hangzhou, China, and his advisor is Dr. Peng Cheng. He is the Director of the Department of Cloud Network Intelligence Operation, Alibaba Cloud. His research interests include large-scale virtual network control, big data analysis, machine learning for networks, and AIOps for networks.

Rong Wen is an Alibaba Cloud Director. His research interests include network virtualization, high-performance networking, and programmable networking.



Jiming Chen (Fellow, IEEE) received the BSc and PhD degrees both in control science and engineering from Zhejiang University. He was a visiting researcher with University of Waterloo from 2008 to 2010. Currently he is professor with College of Control Science and Engineering, Zhejiang University. His research interests include IoT, networked control, and wireless networks.

Shunmin Zhu is currently pursuing the Ph.D. degree in computer science with Tsinghua University, Beijing, China, and his advisor is Dr. Jiahai Yang. He is also the General Manager of the Department of Cloud Network, Alibaba Cloud. He has developed the Alibaba Cloud Virtual Network System "LuoShen," a large-scale cloud network architecture that supports over 20 data centers across the world, and connects tens of millions of virtual machines and all other Alibaba Cloud products. His research interests include large-scale virtual network measurement, large-scale network anomaly detection, cloud network architecture design, and network hardware offloading.