Scalable and Robust East-West Forwarding Framework for Hyperscale Clouds

Qianyu Zhang, Gongming Zhao, *Member, IEEE*, Liguang Xie, *Senior Member, IEEE*, Hongli Xu, *Member, IEEE*, Zhuolong Yu, Yangming Zhao, Chunming Qiao, *Fellow, IEEE*, Liusheng Huang, *Senior Member, IEEE*, and Ying Xiong

***Abstract*— With the broad deployment of distributed applications on clouds, east-west traffic is now dominating the majority of cloud networks. The existing communication solutions are tightly coupled with either the control plane (*e.g.*, preprogrammed model) or the location of compute nodes (*e.g.*, conventional gateway model). As a result, it is difficult to flexibly respond to the rapidly expanding networks and frequent abnormal events (*e.g.*, burst traffic and device failures). Accordingly, they may not provide high-performance east-west forwarding while ensuring scalability and robustness. To address this issue, we design Zeta, a scalable and robust east-west forwarding framework with gateway clusters for hyperscale clouds. Zeta abstracts the traffic forwarding capability as a Gateway Cluster Layer, decoupled from the logic of control plane and the location of compute nodes. Specifically, Zeta adopts gateway clusters to support large-scale networks and cope with burst traffic. Moreover, a transparent Multi IPs Migration is proposed for fast recovery from unpredictable failures. We implement Zeta based on eXpress Data Path (XDP) and evaluate its scalability and robustness through comprehensive experiments with up to 100k container instances. Our evaluation shows that Zeta reduces the 99% RTT by 5.1× in burst video traffic, and reduces the gateway pure recovery delay by 10.8× compared with the state-of-the-art solutions.**

***Index Terms*— East-west forwarding, hyperscale clouds, scalability, robustness.**

I. Introduction

SOFTWARE-DEFINED networking (SDN) separates the control plane and data plane [2], and its concepts and technologies are the cornerstones of current hyperscale cloud networks [3], [4]. With an increasing number of distributed applications on clouds, east-west communication between instances has become the majority load (even up to 75% [5]) in cloud networks [6]. The management and installation methods of flow entries in SDN, including proactive model and reactive model, will significantly affect the forwarding delay and network scale [7]. Similarly, the installation of the forwarding rules is critical for high-performance, scalable and robust east-west communication in cloud networks. However, two factors bring much pressure on cloud networks. First, a hyperscale cloud can accommodate over 100k servers and millions of instances with Pbps bandwidth [8], bringing congestion risks to the network. According to the five-month monitoring of a network with 27.6k servers, there exists at least one highly congested region for more than 56% of uptime, and these congestion regions have a maximum size of 2324 links [9]. Second, containerization leads to centralized startup and short life cycles of instances, which bring great dynamics to the network. For example, Google launches several billion containers per week into Google Cloud [10], [11].

As a result, the east-west forwarding faces several challenges in large-scale and highly dynamic cloud networks. (1) *Scalability.* The expansion of the instances scale in cloud networks leads to a rapid increase in forwarding rules consumption. For example, the control plane will install 487M rules for a preprogrammed network with 40k instances [4], which increases forwarding delay due to the rules lookup. Therefore, installment of numerous rules will limit the scale-up of both single VPC and the whole network. (2) *Robustness.* Although the failure probability of a specific equipment is usually low, network abnormal events in hyperscale clouds are frequent and inevitable, including device failures [12], [13] and burst traffic [14], [15]. They pose severe network congestion/interruption and degrade the tenants' experience. (3) *Latency.* The delays of configuring forwarding rules and establishing/resuming communication are crucial metrics. When instances launch/migrate, some previous solutions require the control plane to inform all relevant hosts and install/update rules, which especially affects short-lived tasks. For example, a function task (*e.g.*, MilliSort and MilliQuery [16]) usually completes in milliseconds, while it may take a few seconds to launch a function instance and establish connection for it.

The existing east-west forwarding solutions in cloud networks are usually divided into two main categories. One is the hardware solutions, such as AWS Nitro System [17], [18], Azure FPGA-based SmartNIC [19], [20], [21], [22] and AliCloud P4-based Gateway [23]. The other is the software solutions, including the preprogrammed model (*e.g.*, VMware

Manuscript received 23 February 2022; revised 25 October 2022; accepted 16 April 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Llorca. Date of publication 8 May 2023; date of current version 19 December 2023. This work was supported in part by the National Science Foundation of China (NSFC) under Grant 62102392, in part by the National Science Foundation of Jiangsu Province under Grant BK20210121, in part by the Hefei Municipal Natural Science Foundation under Grant 2022013, and in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant 2023481. Some preliminary results of this paper were published in the Proceedings of USENIX NSDI 2022 [1]. *(Corresponding author: Gongming Zhao.)*

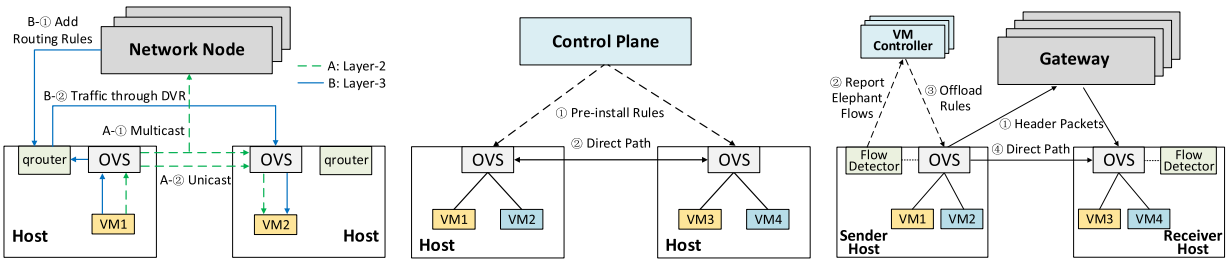
Qianyu Zhang, Gongming Zhao, Hongli Xu, Yangming Zhao, and Liusheng Huang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu 215123, China (e-mail: zqy2019@mail.ustc.edu.cn; gmzhao@ustc.edu.cn; xuhongli@ustc.edu.cn; zhaoym.ustc@gmail.com; lshuang@ustc.edu.cn).

Liguang Xie and Ying Xiong are with Futurewei Technologies Inc., Santa Clara, CA 95050 USA (e-mail: lxie@futurewei.com; yxiong@futurewei.com).

Zhuolong Yu is with the Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218 USA (e-mail: zhuolong@cs.jhu.edu).

Chunming Qiao is with the Department of Computer Science and Engineering, SUNY at Buffalo, Buffalo, NY 14260 USA (e-mail: qiao@buffalo.edu).

Digital Object Identifier 10.1109/TNET.2023.3269772



(a) Neutron Model (b) Preprogrammed Model (c) Gateway Model

Fig. 1. Three Typical East-West Forwarding Models. (a) Neutron model realizes layer-2 communication by learning MAC address and utilizes DVR (qrouters) for layer-3 communication. (b) Preprogrammed model pre-installs all potential rules when launching VMs. (c) Gateway model pre-installs default rules pointing to the gateway on the host. The gateway forwards the header packets, and the direct path rules of elephant flows will be offloaded to the source hosts.

NSX [3], [24]) and the gateway model (*e.g.*, Google Cloud Hoverboard [4]). Considering the high cost and long development cycles of hardware, software solutions have become the preferred choice for many medium-sized cloud providers. Although the control/data plane of existing software solutions is based on SDN, they also face several critical disadvantages (see §II-A for details). First, the preprogrammed model pre-installs numerous rules for VMs and is coupled with the control plane. The conventional gateway model depends on fixed gateways allocated for host zones and is coupled with the location of compute nodes. Hence, they lack the scalability or robustness to adapt to large-scale networks. Second, the existing control loops are complex, which slows the recovery from network abnormal events, including device failure/overload and VM migration.

To overcome the above challenges, we propose a scalable and robust east-west forwarding framework for hyperscale clouds, called Zeta. Zeta abstracts the traffic forwarding capability as a gateway cluster layer, decoupled from the location and logic of other modules. Specifically, Zeta mainly proposes the following innovative designs. (1) Zeta utilizes gateway clusters to improve the fault tolerance of a single gateway and leverages eXpress Data Path (XDP) [25] to accelerate gateway forwarding, thereby enhancing the network scalability and robustness. (2) Zeta adopts the flow table and group table [26] to realize the intra-cluster gateway load balancing. (3) Zeta proposes Multi IPs Migration to achieve gateway fast recovery, which implements failover by migrating the vIPs of the failed gateways. This scheme avoids updating the on-host default rules pointing to the gateways, making failure recovery transparent to hosts/tenants.

The main contributions of this paper are as follows:

- We analyze the pros and cons of existing typical east-west forwarding models for hyperscale clouds and present the design principles of our framework.
- We design a prototype framework, called Zeta, to achieve scalable and robust east-west forwarding for hyperscale clouds. Zeta is publicly available at https://github.com/futurewei-cloud/zeta/.
- We evaluate the robustness and scalability of Zeta with comprehensive experiments. Zeta can reduce the 99% RTT by 5.1× in burst video traffic, and reduce the gateway pure recovery delay by 10.8× compared with the state-of-the-art solutions.

II. Background and Motivation

We will analyze the limitations of three typical east-west forwarding models for hyperscale clouds and motivate our work in this section.

A. Limitations of Prior Works

As an open source cloud computing architecture, OpenStack helps quickly deploy small-scale clouds [29]. As shown in Fig. 1(a), OpenStack Neutron provides the networking capability for the clouds. Specifically, Neutron provides layer-2 networking communication by learning MAC address [27]. When two VMs in the same layer-2 domain communicate for the first time, the source VM will broadcast ARP packets to obtain the MAC address of the destination VM. However, when encountering burst traffic in large-scale networks, it may cause unnecessary layer-2 broadcast flooding, leading to poor robustness and scalability [30]. For layer-3 networking, all the traffic will be routed by specific network node(s) in the initial OpenStack releases. It may suffer the risk of network node(s) failure and high forwarding delay in large-scale networks. To this end, OpenStack has released the Distributed Virtual Router (DVR) since Juno version [28], which can significantly mitigate the robustness and latency issues. However, DVR suffers the oversize routing tables and frequent synchronization problems, which also decrease the network scalability [31]. In general, OpenStack gradually improves forwarding performance through evolutions. But due to the lack of targeted designs for hyperscale clouds, it still faces robustness and scalability issues.

To reduce the forwarding latency between VMs, the preprogrammed model was adopted by many platforms (*e.g.*, VMware NSX [3], [24]), which is similar to the proactive model in SDN. As shown in Fig. 1(b), the control plane pre-installs all potential rules when launching VMs, as it cannot exactly predict which pairs of VMs will communicate. The traffic between VMs will be forwarded directly with low delays. However, the preprogrammed model brings some non-negligible system overhead. First, it will pre-install a quadratic number of rules on hosts, which limits the network scalability. Specifically, in a cloud with h hosts and n VMs, it requires a total of n^2 rules to be pre-installed in the worst case, and launching a new VM requires $2n$ rules to be pre-installed. A massive number of pre-installed rules will slow down the rules lookup and traffic forwarding, thus limiting the network scale. Second, numerous preprogrammed rules seriously delay the VMs deployment/migration. The control plane needs to pre-install/update all potential rules on hosts, which will cause a significant programming delay in communication establishment/recovery. For example, the preprogrammed model takes 74 seconds to install 487M rules for a large network with 10k hosts and 40k VMs [3], [4]. Above system overhead leads to poor scalability of the preprogrammed model, especially in large-scale cloud networks.

TABLE I
COMPARISON OF THE ADVANTAGES AND DISADVANTAGES OF EXISTING MODELS

Models	Robustness		Scalability		Latency		
	Gateway Failure	Burst Traffic	VPC Size	Global Scale	Forwarding	VM Launching	VM Migration
Neutron [27, 28]	✗	✗	✗	✗	✗	✓	✗
Preprogrammed [3, 24]	—	✓	✗	✗	✓	✗	✗
Gateway [4]	✗	✗	✓	✓	✓	✓	✓
Ours: Zeta	✓	✓	✓	✓	✓	✓	✓

To overcome the disadvantages of the former two models, the gateway model on-demand installs rules, and has been widely adopted by cloud providers (*e.g.*, Google Cloud Hoverboard [4]), which is similar to the reactive model in SDN. As shown in Fig. 1(c), the gateway model organizes all servers into host zones. Host zone is a collection of colocated machines with uniform network connectivity, each of which is equipped with a primary gateway and several backups. This model only pre-installs default rules pointing to the gateway on the host's vSwitch. When a new flow arrives, the vSwitch sends the header packets to the gateway according to the default rules. Then, the gateway forwards these packets and offloads direct path rules for elephant flows [4], so that the subsequent packets of those elephant flows will be forwarded to the destination directly.

The gateway model improves the network scalability through on-demand rules offloading. However, it usually allocates a primary gateway to each host zone and may encounter the robustness issues. 1) *Gateway failure.* Although the primary-backup gateway model synchronizes the status in real time to provide disaster tolerance, it still takes a long time to migrate the traffic from the primary gateway to the backup. Specifically, when the primary gateway fails, the default Open vSwitch (OVS) [32] entries of all hosts in the affected host zone need to be updated to point to the backup gateway. In addition, the gateway model will install a default OVS entry for each VPC on the host due to VPC isolation. Considering that there are usually hundreds of hosts in a host zone, and each host may have dozens of VPCs, modifying the default OVS entries of all hosts will cause a significant updating delay. 2) *Burst traffic.* The gateway model usually assigns only one primary gateway to each host zone, so load balancing cannot be directly applied. When a host zone encounters burst traffic, the corresponding primary gateway will be easily overloaded (especially when the control plane cannot detect and offload elephant flows immediately).

B. Our Intuitions

As summarized in Table I, the gateway model combines the advantages of both Neutron and Preprogrammed model in terms of scalability and latency. However, the existing gateway model usually assigns fixed gateway(s) to each host zone. Its gateways incur a high risk of overload/failure under abnormal events, including burst traffic and gateway failures. A natural solution is to deploy multiple primary gateways in a host zone to alleviate the impact of burst traffic or abnormal events. However, the gateways need to be provisioned for peak bandwidth usage, making it difficult to efficiently schedule gateway resources [4].

Another intuitive solution is to organize all gateways into a large virtual cluster to improve disaster tolerance. The new arrival flows will be forwarded to gateways through Equal-Cost Multi-Path (ECMP) [33]. However, once VMs launching/migration occurs, the control plane should notify all gateways to update the forwarding rules, which brings high synchronization overhead on both the gateways and the control plane [34]. For example, assuming that a large datacenter contains 500 gateways and launches 3k containers per second [10], [11]. The controller needs to push 1.5M rules in one second, which poses a severe risk of control plane overload and is not feasible for hyperscale clouds.

In order to integrate the pros of the existing gateway model, but mitigate the cons discussed above, Zeta focuses on improving robustness through two design principles. First, all gateways are divided into several clusters, which effectively improves fault tolerance while reducing the synchronization overhead, as the controller only needs to push latest forwarding rules to the gateways of one cluster every time. Second, we abstract the forwarding function of gateways into the gateway cluster layer. Each VPC allocates a gateway cluster, which decouples the location of gateways and hosts, as the most widely used leaf-spine topology [35] in data centers makes the forwarding delay of gateways predictable and not affected by location. Specifically, the direct communication traffic between two instances needs to pass through at most 3 switches in a two-tier leaf-spine topology. Traffic forwarded by one gateway needs to pass through at most 6 switches.

On the one hand, Zeta achieves gateway cluster load balancing to effectively deal with burst traffic. Specifically, Zeta uses OVS flow tables and group tables to achieve intra-cluster load balancing, and applies VPC-cluster mapping algorithm for inter-cluster load balancing. On the other hand, the independence of Zeta gateway cluster brings flexibility to failure recovery. Zeta adopts the Multi IPs scheme, which is transparent to hosts/tenants to achieve fast recovery from gateway failure/overload/expansion. In addition, we choose XDP as the data plane of Zeta, because of its advantage of obtaining resources on demand and integration with Linux kernel. Based on the above ideas, we design a scalable and robust east-west forwarding framework for hyperscale clouds to support high-performance traffic forwarding.

III. SYSTEM DESIGN

A. Design Goals

Zeta is an east-west forwarding framework with gateway clusters for hyperscale clouds. Our design goals are as follows:

- **Robustness:** High reliability is the core requirement of east-west forwarding, especially for cloud providers. In particular, Zeta focuses on effectively dealing with burst traffic and abnormal events (*e.g.*, gateway failure/overload/expansion), to avoid network congestion/interruption degrading the tenants' experience.
- **Low Latency:** Since east-west traffic is very sensitive to latency, Zeta aims to reduce the traffic forwarding

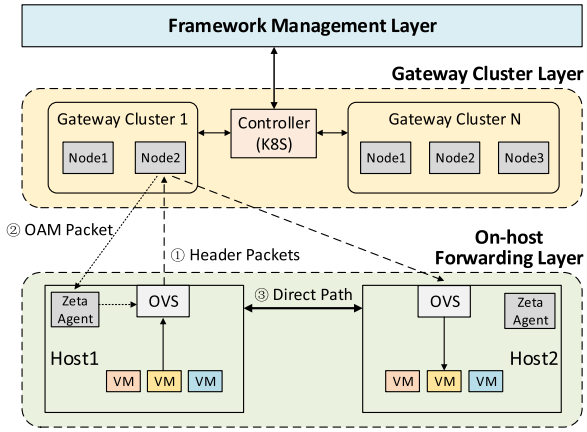


Fig. 2. Zeta Framework Overview. Gateway Cluster provides high-performance traffic forwarding and on-demand rules offloading for tenant instances. On-host Forwarding transmits traffic according to default/direct rules and achieves the intra-cluster gateway load balancing through group tables. Framework Management manages the whole network and further improves the forwarding robustness.

latency through the high-performance in-kernel fast-path. In addition, the lightweight control loop helps reduce the programming delay of VMs launching/migration.

- **Scalability:** With the rapid growth of cloud scale, Zeta should better support large-scale virtual networks up to 100k instances.
- **Compatibility:** Zeta is open source and can also serve as a common hosting platform to integrate customization network functions into the overall virtual networking.

B. System Overview

As shown in Fig. 2, to realize the above design goals, we propose an efficient east-west forwarding framework, called Zeta, which consists of three core modules: Gateway Cluster, On-host Forwarding and Framework Management.

Gateway Cluster Layer establishes a forwarding network based on VXLAN tunnel [36]. It leverages XDP to provide high-performance traffic forwarding and on-demand rules offloading for tenant instances (§IV-B). The application of gateway cluster ensures better scalability and robustness. Gateways detect the elephant flows and send OAM (Operations, Administration and Maintenance) packets to the source hosts, which contain direct path rules (§IV-C). We further design the timeout mechanism of eBPF map for flow statistics (§IV-D). In addition, Zeta adopts *Multi IPs Migration* to achieve fast recovery from gateway failure/overload/expansion, which makes failure recovery transparent to hosts/tenants (§IV-E).

On-host Forwarding Layer transmits traffic according to the rules on OVS. Before deploying a new VPC, a default rule will be pre-installed on the host, which consists of a flow entry and a group entry to achieve the intra-cluster gateway load balancing (§V-A). When two VMs communicate for the first time, the header packets will be sent to a specific gateway according to the default rule. Each host deploys a *Zeta Agent*, which is responsible for parsing OAM packets and installing the direct path rules on the on-host OVS. In addition, the lightweight control loop based on *Zeta Agent* can make a quick response to network adjustments, such as passive instance migration (§V-B).

Framework Management Layer manages the whole network and further enhances the robustness of gateway clusters.

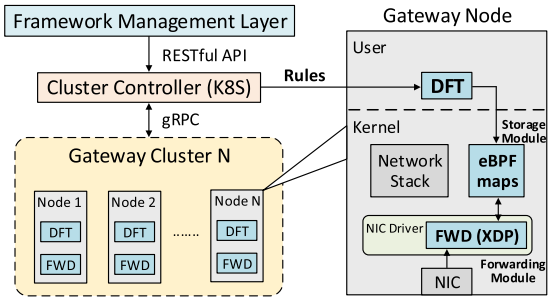


Fig. 3. Illustration of Gateway Cluster Design. The left plot is the overview of gateway cluster and the right plot is the implementation details of XDP-based gateway.

When Zeta is initialized, the management layer will determine the VPC-cluster mapping for inter-cluster load balancing (§VI-A). To deal with the abnormal events and traffic dynamics, the *Multi IPs Scheduler* will dynamically adjust the configurations (*e.g.*, multi IPs allocation and cluster partition), thereby avoiding overload of partial clusters for better robustness (§VI-B).

IV. Gateway Cluster Design

A. Gateway Cluster Overview

Zeta Gateway Cluster establishes a VXLAN-based forwarding network. Specifically, it provides high-performance traffic forwarding and on-demand rules offloading for tenant instances with scalability and robustness guarantee. As shown in the left plot of Fig. 3, Gateway Cluster Layer consists of a cluster controller and several gateway clusters.

Cluster Controller contains management and scheduling logic for gateway clusters. First, it interacts with the Framework Management Layer through its Northbound RESTful API. Second, it manages the gateway clusters and maintains the gateways load balancing through its Southbound API based on gRPC [37]. Cluster Controller is hosted on a Kubernetes cluster. We directly adopt Kubernetes's own health check and instance automatic recovery strategy to ensure the reliability of the control plane.

Gateway Clusters constitute the data plane of the forwarding network. We divide all gateways into several clusters to achieve the robust gateway forwarding. In practice, each cluster consists of several isomorphic gateways, which store the same forwarding rules to collectively provide traffic forwarding and rules offloading services for tenant instances. Each gateway contains the Forwarding Module (FWD) and the Distributed Flow Table Module (DFT). Specifically, FWD forwards the packets to the destination hosts and offloads direct forwarding rules to the source hosts for those elephant flows. DFT is a lightweight key-value store, which maintains a consistent forwarding table on each gateway of a cluster. When the forwarding table changes (*e.g.*, instances launching/migration), the cluster controller will push the latest rules to each gateway within the corresponding cluster. In addition, there is no state synchronization among gateways (in §IV-C).

B. XDP-Based Traffic Forwarding

The forwarding module of a Zeta gateway is implemented based on XDP [25] to improve the forwarding performance and reduce the transmission latency. XDP is a high-performance in-kernel fast data path, which mounts programs on the NIC driver and realizes pre-stack packet

processing [25], [38]. On arriving at the gateway, the packets will be processed and forwarded before the kernel network stack, which avoids kernel stack overhead and reduces forwarding delay. In addition, XDP adopts interrupt mode and obtains CPU resources on demand according to the traffic bandwidth [25]. Though XDP is not the first mover in this area, we choose XDP as the data plane of Zeta, because of its on-demand resource acquisition and integration with Linux kernel. As illustrated in the right plot of Fig. 3, we converge the forwarding, computing and storage functions together, which eliminates the overhead of network stack processing [39], [40].

Forwarding Module works at the NIC driver and can directly operate on raw Ethernet frames. The workflows of XDP-based forwarding program are as follows: (i) Receiving header packets of the source instance from the NIC RX buffer. (ii) Obtaining the forwarding rule of the target instance by querying the storage module, that is, determining the destination host of the traffic. (iii) Parsing the protocol field of VXLAN inner packets. ARP messages will be directly responded to the source instance, while other types of packets will be forwarded to the destination. (iv) Sending OAM (Operations, Administration and Maintenance) packets containing direct rules to the source hosts for the elephant flows.

Storage Module consists of several eBPF maps [41], [42]. These maps are key-value stores [43] that serve as the data channel between DFT and FWD. The forwarding module will also cache the real-time information of flows in eBPF maps. For example, FWD will count the OAM packets generated for each flow to avoid repeated offloading.

C. Gateway Flow Detection

In order to further reduce the rules stored on the hosts, so as to save memory and reduce the forwarding delay caused by rules lookup. Zeta adopts XDP's high-performance packet processing features to detect elephant and mice flows on the gateway, which can improve the efficiency of the detection program and the system's robustness. When encountering burst traffic generated by a simultaneous batch of workloads (*e.g.*, MapReduce [44]), the on-host flow detection program of existing gateway model may be overloaded, as its host agent is usually equipped with limited resources, *e.g.*, 1 CPU core and 1.5GB memory [4]. In contrast, the additional overhead of detecting elephant flows is almost negligible for the XDP-based gateways of Zeta while forwarding traffic.

When traffic arrives at the XDP forwarding module, it will accumulate the total size of each flow in a certain period and store the records in an eBPF map. If the cumulative size of a flow exceeds the threshold (*e.g.*, 20kbps [4]) before the next period, it will be identified as an elephant flow and offloaded to the source hosts. Each flow is only sent to a specific gateway according to the 5-tuple hash (in §V-A), which avoids synchronization of flow size statistics among gateways. In addition, Zeta will monitor the gateway load. When a gateway's CPU or memory utilization reaches the threshold (*e.g.*, 80%), the gateway will pause the elephant flows detection and offload direct rules for all flows.

D. Timeout Mechanism Design of eBPF Map

We use the eBPF map to store the periodic statistics of flows for OAM counter and flow detection. Because the eBPF map does not provide the timeout mechanism, that is, an entry cannot automatically reinitialize after a period, we implement this mechanism by ourself. An intuitive solution for the timeout mechanism is to maintain an extra map for each map to record the expiration time of entries. However, this approach incurs several additional map read/writes, leading to throughput degradation as the number of entries increases, which has been verified by the experiments in §VIII-B. We further optimize the timeout mechanism of eBPF map with one read/write so reduce the impact of the number of entries on the forwarding performance.

Specifically, we maintain a `flow` map to store various periodic flow statistics and corresponding initialization timestamps. For each arriving packet, the forwarding program will only read and write the `flow` map once according to the 5-tuple of this flow. For flow detection, the timeout mechanism of the `flow` map entry is as follows: (i) If the flow entry is not queried, the program will initialize an entry for this flow, and record the current time stamp and packet size of this flow in the entry. (ii) If the flow entry is queried, the program will update the entry for flow statistics. Specifically, if the difference between the current time and the time stamp recorded in the entry exceeds the threshold, the program will update the time period, and the entry will timeout and initialize. Otherwise, the flow entry is not expired, the program will execute the flow statistics and update the entry.

E. Dealing With Failures Through Multi IPs

The number of gateways in a cluster will change dynamically due to gateway failures and scaling requirements, and the hash modulo of the default rule will change accordingly (*i.e.*, group entry buckets in §V-A). Therefore, we have to modify all the installed default rules associated with the updated cluster. To this end, massive affected hosts need to be informed, which leads to heavy notification overhead and unacceptable delay [45]. To address this issue, we design the *Multi IPs Migration*. Briefly, each gateway node is logically assigned multiple virtual IPs (vIPs), and the vIPs can be reallocated among nodes. Tenant traffic is bound to vIPs and decoupled from gateways.

The feature of XDP working in the layer-2 networking inspires a solution of gateway failure recovery. We propose the *Multi IPs* scheme to achieve fast failure recovery. Specifically, the cluster controller maintains a *Multi IPs Mapping Table*. When a gateway cluster is initialized, each gateway node in the cluster will be allocated several logical virtual IP-MAC pairs, and send RARP packets [46] to add MAC table entries on the connected Top-of-Rack (ToR) switch(es). It should be noted that these vIPs and vMACs are not actually configured in the gateways' NIC, as XDP program can directly operate on the raw Ethernet frames. When a gateway fails, the cluster controller will reassign the logical vIP-vMAC pairs of the failed gateway to other healthy gateways in the cluster. Since the forwarding rules maintained by each gateway in a cluster are consistent, there is no synchronization overhead/delay among gateways during failure recovery. Next, the healthy gateways that have obtained migrated vIP-vMAC pairs will utilize RARP to inform the connected switch(es) to update MAC address table. Then, the packets from instances can be correctly forwarded to healthy gateways.

Fig. 4 illustrates an example of fast recovery through *Multi IPs Migration*. Initially, Gateway Cluster 1 contains three

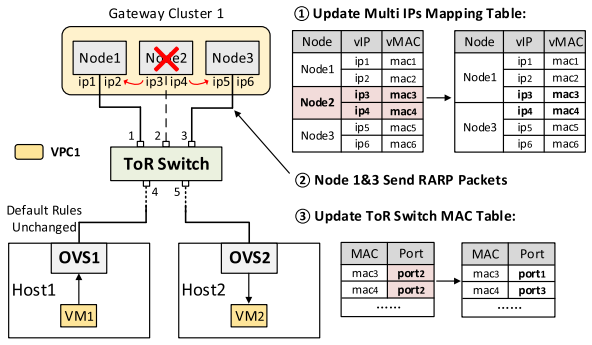


Fig. 4. Dealing with Gateway Failures through *Multi IPs*. When Node2 fails, the cluster controller first updates the mapping table to reassign the vIP-vMAC pairs to healthy gateways (*i.e.* Node 1&3). Then Node 1&3 send RARP packets to update the MAC entries on the connected switch. The recovery scheme avoids modifying the default OVS rules on hosts.

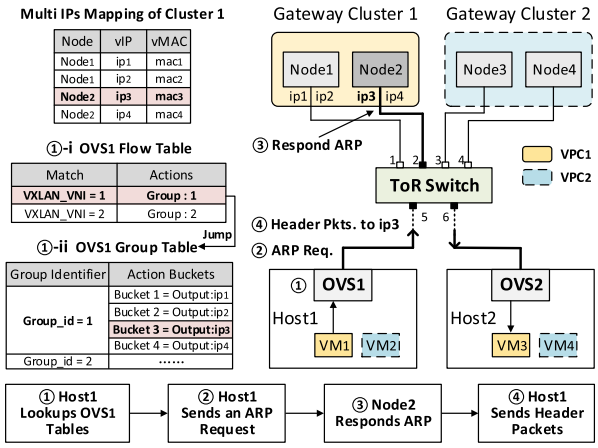


Fig. 5. Illustration of Interaction between Flow Table and Group Table. When VM1 belonging to VPC1 communicates with VM3 for the first time, Host1 lookups the OVS1's default tables, and the default gateway IP of VM1's flow is ip3. Then, the source host uses the ARP protocol to obtain that the target gateway corresponding to ip3 is Node2. Finally, Host1 sends the header packets of VM1 to Node2.

gateway nodes, each of which is assigned with two vIP-vMAC pairs, as shown in the *Multi IPs Mapping Table*. When Node2 fails, the cluster controller will update the mapping table, ip3-mac3 and ip4-mac4 originally assigned to the Node2 are reassigned to Node1 and Node3 respectively. Next, Node1 and Node3 utilize the RARP protocol to update the MAC address table of the connected ToR switch, so that the packets toward the failed Node2 will be immediately diverted to the healthy nodes. As a result, the failure recovery is transparent to hosts/tenants without modifying any default OVS entry or on-host ARP cache that involves the failed gateway(s), which significantly reduces the recovery delay and enhances the system robustness. According to the experiments in §VIII-C.2, Zeta reduces the gateway pure recovery delay from 62ms to 5.5ms.

Discussion. Of course, Multi IPs Migration can also be applied to the existing gateway model. That is, when the primary gateway fails, its IP will be migrated to the backup gateway. In this case, the existing gateway model is a special case of Zeta gateway cluster, that is, each cluster has only one active gateway. However, Zeta has additional advantages in adopting *Multi IPs*: (1) Intra-cluster load adjustment and (2) Rapid cluster scaling (covered in §VI-B).

V. On-Host Forwarding Design

A. Load Balancing Through Group Tables

This section elaborates on the designs of default entries to achieve intra-cluster gateway load balancing. In order to realize the decoupling of gateway cluster and location (*i.e.*, host or host zone), we construct default rules in VPC granularity. Thus, when launching a new VPC on a compute node, the default rule of this VPC will be pre-installed by Zeta Agent on the on-host OVS.

To achieve the gateway load balancing within a cluster, we utilize the flow table and group table of on-host OVS to orchestrate the gateway clusters. In brief, each entry of the group table points to a cluster, and the buckets in each group entry specify the gateway nodes in this cluster. Specifically, when the header packet of a flow reaches OVS, it first matches the flow entry and jumps to a group entry according to the VPC identifier (`VPC_id`) so that the target cluster for this flow is determined. The VPC-cluster mapping algorithm will be elaborated in §VI-A. Then, the packet will be hashed to a bucket in the group entry according to the 5-tuple, which determines the destination vIP for this flow. Finally, the source host adopts the ARP protocol to obtain the target gateway corresponding to the destination vIP. As the group table selects the target gateway based on the 5-tuple hash, load balancing within a gateway cluster can be guaranteed.

We give an example in Fig. 5 to illustrate the intra-cluster gateway load balancing with the flow table and group table. Assuming that VM1 belonging to VPC1 communicates with VM3 for the first time. When the header packet arrives at the OVS of Host1, the OVS first matches the flow entry with `VXLAN VNI=1` and jumps to the group entry with `Group_id=1`. Each bucket in a group entry corresponds to the IP address of a gateway node in the cluster, and the packet will be hashed to a bucket according to its 5-tuple. In our example, the packet is hashed to bucket3, that is, the destination vIP of the packet is ip3. Then, Host1 sends an ARP request to obtain the vMAC of ip3, and Node2 in Cluster1 responses the ARP, that is, the packets will be forwarded to the gateway Node2 by default. Finally, Host1 sends the header packets of VM1 to Node2, and the gateway will forward these packets and offload a direct rule to the source Host1.

B. Lightweight Control Agent

The lightweight control loop based on *Zeta Agent* can effectively reduce the recovery latency of the passive instance migration, such as Kubernetes Pod Eviction [47]. Assuming that compute nodes deploy a Kubernetes cluster, when a host is out of resources, the Kubernetes scheduler [48] will migrate the relevant pod(s) to other host(s). Conventionally, Kubernetes will not inform gateways or on-host agent of pod migration actively. The agent of existing gateway model needs to poll Kubernetes database (*e.g.*, Etcd [49]) to obtain the latest pod location. Therefore, the hosts cannot update the offloaded rules immediately. The traffic is still forwarded to the former destination hosts, which results in a network interruption between the affected pods.

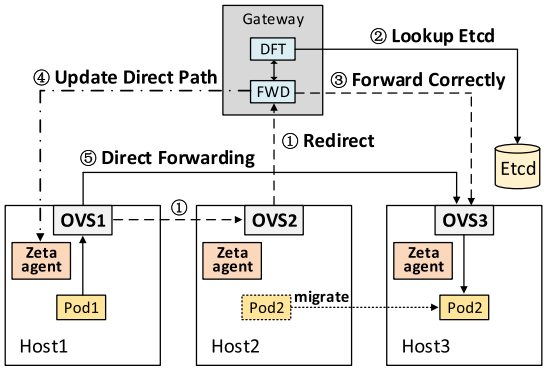


Fig. 6. Lightweight Control Agent on compute nodes. When Pod2 is migrated, the flows sent to Pod2 will be redirected to gateway. The gateway forwards the flows and queries the database, then updates the direct path on the source host.

Three steps are required to recover the communication of migrated pods: (i) Obtaining the latest forwarding rules. (ii) Redirecting the packets toward the migrated pods to the correct destination. (iii) Updating the direct rules on the source hosts. We hope that Zeta reduces the recovery delay of passive instance migration and *Zeta Agent* remains lightweight to occupy fewer host resources. Thus, instead of directly implement above three steps on agent, Zeta Agent redirects the traffic destined for the migrated pods to gateways. The gateways query the latest rules and forward the traffic to the correct destination.

As illustrated in Fig. 6, when Pod2 is migrated, the Zeta Agent on Node2 will install an entry on OVS2 to redirect all packets toward the Pod2 to the gateway. FWD on Zeta gateway recognizes the redirected packets and reports their destinations to DFT. DFT queries the latest location information of Pod2 from Kubernetes database and updates the rules in FWD. Then, FWD will forward the redirected packets to the correct destination Node3, and send OAM packets to the source Node1. Finally, the Zeta Agent on Node1 will update the direct forwarding rule to Pod2.

VI. Framework Management Design

A. Gateway Cluster Mapping

When Zeta initializes, the management layer will determine the VPC-cluster mapping for inter-cluster load balancing.

1) System Model: In the Zeta framework, we use $C = \{c_1, c_2, \ldots, c_n\}$ to denote the gateway clusters, where $n = |C|$ is the number of clusters. For each gateway cluster c, its forwarding capacity is denoted as $B(c)$. We denote $V = \{v_1, v_2, \ldots, v_{|V|}\}$ as the VPC set. Let $T = \{t_1, t_2, \ldots, t_d\}$ denote the tenants set, where $d = |T|$ is the number of tenants in the cloud. Each tenant $t \in T$ consists of a VPC set $V_t = \{v_1^t, v_2^t, \ldots, v_{|V_t|}^t\}$. Obviously, $V = V_1 \cup V_2 \ldots \cup V_d$. Moreover, the traffic demand of each VPC is denoted as $f(v)$.

2) Problem Formalization: We define the gateway clusters mapping (GCM) problem in the Zeta framework. To enhance the system robustness and improve the QoS. We need to consider the following two constraints. (1) ***VPC Constraint***. A VPC will be mapped to one and only one gateway cluster, as all the vIPs of a group entry belong to the same cluster (§V-A). (2) ***Tenant Constraint***. We limit the number of gateway clusters that each tenant can be mapped to. For security reasons, we do not expect that burst/malicious traffic from a single tenant will affect all gateway clusters.

Moreover, we use binary $x_v^c \in \{0, 1\}$ to denote whether a VPC $v \in V$ is mapped to a gateway cluster $c \in C$ or not. Let binary $y_t^c \in \{0, 1\}$ represent whether the gateway cluster $c \in C$ is assigned the VPCs belonging to tenant $t \in T$ or not. The objective of GCM is to achieve the load-balancing among all gateway clusters. We formulate GCM as follows:

$$\begin{aligned} \min \quad & \lambda \\ S.t. \quad & \begin{cases} \sum_{c \in C} x_v^c = 1, & \forall v \in V \\ \sum_{v \in V} x_v^c \cdot f(v) \le \lambda B(c), & \forall c \in C \\ x_v^c \le y_t^c, & \forall v \in V_t, c \in C, t \in T \\ \sum_{c \in C} y_t^c \le k, & \forall t \in T \\ x_v^c \in \{0, 1\}, & \forall v \in V, c \in C \\ y_t^c \in \{0, 1\}, & \forall t \in T, c \in C \end{cases} \end{aligned} \tag{1}$$

The first set of equations means that all traffic of a VPC will be forwarded to one gateway cluster by default. The second set of inequalities describes the traffic load on each gateway cluster, where $\lambda \in [0, 1]$ represents the load balancing factor. The third set of inequalities indicates that the tenant t is mapped to gateway cluster c only if VPC(s) of tenant t is processed by cluster c. The fourth set of inequalities represents the *Tenant Constraint*, that is, the VPCs of a tenant will be mapped to at most k gateway clusters. Our objective is to achieve the load balancing among all gateway clusters, *i.e.*, minimizing the load balancing factor λ.

Theorem 1: The GCM problem is NP-hard.

Proof: We consider a simplified version of GCM problem without the *VPC constraint*. Then the simplified GCM problem becomes a Parallel Machine Scheduling (PMS) problem [50]. Since the PMS problem is NP-hard, the GCM problem is NP-Hard too. □

3) Rounding-Based Mapping Algorithm: To solve the problem in Eq. (1), we propose a rounding-based gateway cluster mapping (RGCM) algorithm for the GCM problem, described in Alg.1, which includes two steps. The first step is to construct linear programming as relaxation of GCM (LP-GCM). Specifically, LP-GCM assumes that the traffic of each VPC can be splittable and forwarded to multiple gateway clusters. Since LP-GCM is a linear programming, we can obtain the fractional solutions $\{\widetilde{x}_v^c\}$ and $\{\widetilde{y}_t^c\}$ by a convex programming solver, such as Gurobi [51]. The optimal fractional result is denoted as $\widetilde{\lambda}$.

The second step is to derive an integer solution, denoted as $\{\widehat{x}_v^c\}, \{\widehat{y}_t^c\}$ randomized rounding [52]. (1) For each individual tenant $t \in T$ and each gateway cluster $c \in C$, $\widehat{y}_t^c$ is set to 1 with the probability of $\widetilde{y}_t^c$ (lines 5-10). Here $\widehat{y}_t^c = 1$ means that the traffic of tenant t will be processed by gateway cluster c. If the set of optional gateway clusters for the tenant t is empty after one round of traversal, we will repeat the traversal

Algorithm 1 RGCM: Rounding-Based Mapping Algorithm

1: **Step 1: Solving the Relaxed RGCM Problem**
2: Construct a linear programming of the problem formalized in Eq. (1)
3: Obtain the optimal fractional solutions $\{\widetilde{x}_v^c\}, \{\widetilde{y}_t^c\}$
4: **Step 2: Gateway Cluster Selection for Each VPC**
5: **for** Each tenant $t \in T$ **do**
6: Define $C_t = \{c | \widehat{y}_t^c = 1\}$
7: **repeat**
8: **for** Each cluster $c \in C$ **do**
9: Set $\widehat{y}_t^c = 1$ with the probability of $\widetilde{y}_t^c$
10: **until** $|C_t| > 0$
11: **for** Each VPC $v \in V$ **do**
12: Define $C_t = \{c | \widehat{y}_t^c = 1, v \in V_t\}$
13: Set $\underset{c \in C_t}{\mathbf{Pr}}[\widehat{x}_v^c = 1] = \frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}$
14: Assign an interval of length $\frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}$ to each cluster $c \in C_t$
15: Generate a random number ξ between $[0, \sum_{c \in C_t} \frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}]$
16: One cluster is assigned to VPC v according to the interval where ξ is located

of the gateway clusters. (2) After variables $\{\widehat{y}_t^c\}$ have been determined, we will assign a default gateway cluster to each VPC $v \in V$ (lines 11-16). RGCM decides the VPC-cluster mapping by rounding variable $\{\widetilde{x}_v^c\}$. We first define the set of clusters that are available to the VPC v belonging to tenant t, as C_t, *i.e.*, $C_t = \{c | \widehat{y}_t^c = 1, v \in V_t\}$. The variable $\widehat{x}_v^c$ is set to 1 with the probability of $\frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}$, *i.e.*, the probability that cluster $c \in C_t$ assigned to VPC v is $\frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}$. We then assign an interval of length $\frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}$ to each cluster $c \in C_t$, and generate a random number ξ between $[0, \sum_{c \in C_t} \frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}]$. Finally, one cluster is assigned to VPC v according to the interval where ξ is located.

4) Performance Analysis: This section will give the approximate ratio of tenant constraint and VPC constraint, and the analysis of load balancing performance. We first introduce two well-known probability theory lemmas:

Lemma 2 (Chernoff Bound): Given n independent variables: $x_1, x_2, \ldots, x_n$, where $\forall x_i \in [0, 1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} x_i]$. Then, $\mathbf{Pr}[\sum_{i=1}^{n} x_i \geq (1+\epsilon)\mu] \leq e^{\frac{-\epsilon^2 \mu}{2+\epsilon}}$, where ϵ is an arbitrarily positive value.

Lemma 3 (Union Bound): Given a countable set of n events: $A_1, A_2, \ldots, A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup \ldots \cup A_n) \leq \sum_{i=1}^{n} \mathbf{Pr}(A_i)$.

Analysis of the Tenant Constraint: The first step of RGCM will obtain the fractional solution $\{\widetilde{y}_t^c\}$ of the relaxed RGCM problem. With the randomized rounding, $\widehat{y}_t^c$ is set as 1 with the probability of $\widetilde{y}_t^c$. Thus, the expectation that gateway cluster c is assigned as the alternative default gateways of the tenant t is given by:

$$\mathbb{E}[\widehat{y}_t^c] = \mathbf{Pr}[\widehat{y}_t^c = 1] = \widetilde{y}_t^c \tag{2}$$

Thus, the expected number of gateway clusters allocated to the tenant t is given by:

$$\mathbb{E}[\sum_{c \in C} \widehat{y}_t^c] = \sum_{c \in C} \mathbb{E}[\widehat{y}_t^c] = \sum_{c \in C} \widetilde{y}_t^c \leq k \tag{3}$$

Theorem 4: With the rounding-based mapping algorithm, the number of gateway clusters that allocated to the tenant t will not exceed k by a factor of $\frac{3 \ln d}{k} + 3$ with a high probability of $1 - \frac{1}{d^2}$, where d denotes the number of tenants.

Proof: For each tenant t, $\widehat{y}_t^c \in \{0, 1\}$ are independent variables with the expectation value $\mathbb{E}[\sum_{c \in C} \widehat{y}_t^c] \leq k$. According to Lemma 2, we have:

$$\mathbf{Pr}[\sum_{c \in C} \widehat{y}_t^c \geq (1+\epsilon)k] \leq e^{\frac{-\epsilon^2 k}{2+\epsilon}} \tag{4}$$

We assume that

$$e^{\frac{-\epsilon^2 k}{2+\epsilon}} \leq \frac{1}{d^3}, d = |T| \tag{5}$$

which means that the probability bound in Eq. 5 goes quickly to zero as the number of tenants d increases. To hold this, ϵ should satisfy:

$$\epsilon \geq \frac{3 \ln d + \sqrt{9 \ln^2 d + 24 k \ln d}}{2k} \tag{6}$$

If we pick $\epsilon = \frac{3 \ln d}{k} + 2$, the above inequality holds. In other words, we have:

$$\mathbf{Pr}[\sum_{c \in C} \widehat{y}_t^c \geq (1+\epsilon)k] \leq \frac{1}{d^3}, \epsilon = \frac{3 \ln d}{k} + 2 \tag{7}$$

Finally, we guarantee the upper bound on the probability that the number of gateway clusters that allocated to a tenant is violated by Lemma 3:

$$\begin{aligned} &\mathbf{Pr}[\bigvee_{t \in T} \sum_{c \in C} \widehat{y}_t^c \geq (1+\epsilon)k] \\ &\leq \sum_{t \in T} \mathbf{Pr}[\sum_{c \in C} \widehat{y}_t^c \geq (1+\epsilon)k] \\ &\leq d \cdot \frac{1}{d^3}, \epsilon = \frac{3 \ln d}{k} + 2 \end{aligned} \tag{8}$$

Therefore, the number of gateway clusters that allocated to tenant t will not exceed k by a factor of $1 + \epsilon = \frac{3 \ln d}{k} + 3$ with a high probability of $1 - \frac{1}{d^2}$. □

Analysis of the VPC Constraint: The probability that gateway cluster b is allocated to the VPC v is given by:

$$\begin{aligned} \mathbf{Pr}[\widehat{x}_v^c = 1] &= \mathbf{Pr}[\widehat{x}_v^c = 1 | y_t^c = 1] \mathbf{Pr}[y_t^c = 1] \\ &\quad + \mathbf{Pr}[\widehat{x}_v^c = 1 | y_t^c = 0] \mathbf{Pr}[y_t^c = 0] \\ &= \frac{\widetilde{x}_v^c}{\widetilde{y}_t^c} \cdot \widetilde{y}_t^c = \widetilde{x}_v^c \end{aligned} \tag{9}$$

The expected number of gateway clusters allocated to the VPC v is given by:

$$\begin{aligned} \mathbb{E}[\sum_{c \in C} \widehat{x}_v^c] &= \sum_{c \in C} \mathbb{E}[\widehat{x}_v^c] = \sum_{c \in C} \mathbf{Pr}[\widehat{x}_v^c = 1] \\ &= \sum_{c \in C_t} \mathbf{Pr}[\widehat{x}_v^c = 1] + \sum_{c \notin C_t} \mathbf{Pr}[\widehat{x}_v^c = 1] \\ &= 1 + 0 = 1 \end{aligned} \tag{10}$$

where the last equation holds according to the second step of RGCM, that selects a default cluster from C_t for VPC $v \in V$ with the probability of $\frac{\widetilde{x}_v^c}{\widetilde{y}_t^c}$. Therefore, the RGCM algorithm will allocate only one gateway cluster to each VPC with the *VPC constraint* guarantee.

Load Balancing Performance Analysis: We compute the expected forwarding load of gateway clusters. We bound the probability with which the forwarding load of gateway clusters will be violated. First, we define a variable l_v^c as the forwarding load of the cluster $c \in C$ allocated to VPC $v \in V$:

$$l_v^c = \begin{cases} f(v), & with\ the\ probability\ of\ \frac{\widetilde{x}_v^c}{\widetilde{y}_t^c} \\ 0, & otherwise \end{cases} \tag{11}$$

The expected forwarding load of the cluster c is:

$$\mathbb{E}[\sum_{v \in V} \widehat{l}_v^c] = \sum_{v \in V} \mathbb{E}[\widehat{l}_v^c] = \sum_{v \in V} f(v) \cdot \widetilde{x}_v^c \leq \widetilde{\lambda} \cdot B(c) \tag{12}$$

To obtain the relationship between load variables and the optimal result, we define the following variable:

$$\beta = \frac{\widetilde{\lambda} \cdot B(c)}{\max_{v \in V} f(v)} \tag{13}$$

According to Theorem 4 and Eq. 12, we can prove the following theorem for load balancing factor.

Theorem 5: The rounding-based mapping algorithm achieves the load balancing factor at most $\frac{3 \ln d}{\beta} + 3$ times worse than the optimal result with a high probability of $1 - \frac{1}{d^2}$, where d denotes the number of tenants.

Since the proof of the above theorem is similar to Theorem 4, we will omit the detailed proof here.

According to the above analysis, we can conclude that the approximate factors in Theorems 4-5 are *bi-criteria approximations* of the *Tenant Constraint* and the load balancing factor. In many practical cases, these factors are constant. For example, assuming that a large datacenter contains 500 gateways and 10,000 tenant, and the forwarding capacity of each gateway is 40Gbps. We divide the 500 gateways into 50 clusters, and the forwarding capacity of each cluster is 400Gbps, *i.e.*, $B(c) = 400$. According to the practical flow traces [53], the maximum traffic of a VPC may reach 10Gbps. We then set the *Tenant Constraint* k as 20 and load balancing factor $\widetilde{\lambda}$ as 0.6. Under this case, we have $\beta = \frac{0.6 \times 400}{10} = 24$, and $\ln d = \ln 10,000 \approx 9.2$. The approximation factors for the *Tenant Constraint* and the load balancing factor are 4.4 and 4.2, respectively.

B. Multi IPs Scheduler

The *Multi IPs Scheduler* executes the IPs migration scheme proposed in §IV-E. It dynamically updates the IPs allocations to eliminate the overload of gateway clusters caused by the burst traffic and abnormal events. In practice, when a gateway exceeds the load threshold (*e.g.*, 80%), it will immediately report such overload to the control plane. Then the *Multi IPs Scheduler* will perform the following two steps:

Step 1: Intra-Cluster Load Adjustment. The scheduler first sorts all gateways of a cluster in the descending order of their load. Next, the scheduler attempts to migrate a vIP-vMAC pair from the overloaded gateway to the gateway with the lightest load, and re-sorts gateways' load. Then, the scheduler will repeat above IPs migration and gateway sorting procedure until none of the gateways in the cluster is overloaded. If we cannot eliminate the overloaded gateways with step 1, the scheduler will go to step 2.

Step 2: Cluster Scaling. If a cluster cannot eliminate overload through internal load adjustment, *e.g.*, a legitimate VPC has burst traffic. The scheduler will migrate gateways from other clusters to this cluster or expand new gateways for this cluster. The scheduler first sorts all the clusters by their average load in the descending order and attempts to reassign a gateway from the least loaded cluster to the overloaded cluster. We can utilize *Multi IPs Migration* to achieve rapid gateway migration among clusters. However, if the gateway migration causes overload risk to the source cluster, the scheduler will directly expand the overloaded cluster with a new gateway.

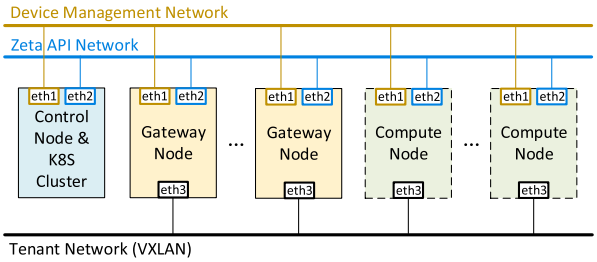


Fig. 7. Best practice of zeta deployment.

VII. Implementation

We implement Zeta based on Linux 5.4 kernel. The Cluster Controller includes 3k lines of Python code, the XDP-based gateway forwarding function includes 4.5k lines of C code, and the Zeta Agent includes 2k lines of C++ code.

Zeta is usually deployed as two self-contained parts: (i) One Kubernetes micro-service for Cluster Controller services; (ii) One Gateway Cluster for Zeta data plane, which is based on physical machines in production environment.

Fig. 7 illustrates the best practice of Zeta deployment, which includes a control node, several gateway nodes and compute nodes. The leftmost control node deploys the management service of the cloud platform and Kubernetes cluster hosting Zeta Controller. The middle ones are gateway nodes, each of which deploys DFT and FWD modules. The `eth1` interfaces of all nodes access the Device Management Network. In addition, we use separate interfaces for the Zeta API Network and Tenant Network, which prevents massive tenants' traffic from blocking the control messages. The Zeta API Network is responsible for sending the operation instructions and reporting status information, including OAM packets, IPs allocation/migration policies and gateways' load status. The Tenant Network transmits the east-west traffic through the VXLAN tunnel [36] for tenant instances.

VIII. Evaluation

We first conduct an ablation analysis to measure the performance of Zeta. We then test the robustness of Zeta under burst

traffic and abnormal events. Finally, we evaluate the scalability of Zeta in public and private cloud scenarios.

A. Experimental Setting

Testbed Setups. We use 23 servers to build the testbed, all running Ubuntu 18.04 with Linux kernel 5.4. Considering our limited number of servers, we deploy KVM-based gateways on several physical machines to simulate gateway clusters. In addition, we launch a large number of container instances on each compute node to evaluate scalability, because of limited number of compute nodes. The scalability in this paper mainly refers to the instance scale, instead of the host scale, as the forwarding rules stored in the gateways and the tenant traffic depend on the instance scale.

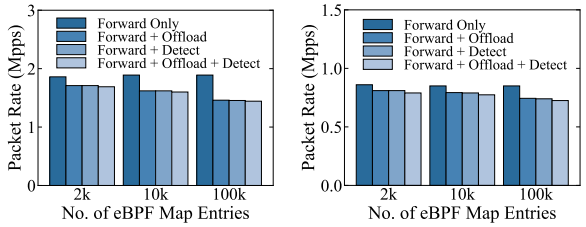
Specifically, 20 servers are used as hosts, each equipped with dual 22-core Intel Xeon 6161 CPUs, 640GB memory and an Intel XL710 40GbE NIC. The other 3 servers are used to deploy gateway clusters, each equipped with dual 16-core Intel Xeon E5-2697A CPUs, 256GB memory and an Intel XL710 40GbE NIC. We deploy a total of 45 KVM-based gateways on the 3 physical gateway machines. Each KVM-based gateway is equipped with 4 vCPUs and 16GB memory. For Zeta, we divide the 45 gateways into 10 clusters, and map VPCs to gateway clusters with the RGCM algorithm to achieve load balancing among clusters.

Moreover, according to the empirical data in [4], we set the rules offloading threshold to 20kbps on the gateway.

Benchmarks. We compare the robustness and scalability of Zeta with other three typical frameworks. The first framework is the conventional gateway model [4], called GWZone, and its gateway is modified based on the implementations of Zeta's gateway. The number of gateways in Zeta is the same as that of main gateways in GWZone, but their gateway allocation methods are different. We divide the on-host instances into 45 parts in sequence, each part is regarded as a host zone, and each host zone is assigned a main gateway. Unlike Zeta, GWZone detects elephant flows on compute nodes. When GWzone faces gateway failure, it will update the default entries on affected hosts and migrate traffic to the backup gateways. We equip GWZone with 9 additional backup KVM-based gateways. As the backup gateways only consume ∼0.1 vCPU and ∼2GB memery in standby, they will not affect the performance of the primary gateways. The second one is the OpenStack Neutron [27], which provides layer-2 networking communication by learning MAC address. The third one is the Preprogrammed model, which is a simplified implementation of VMWare NSX [3], [24] as it is not open source. This model will pre-install all potential rules before launching VMs.

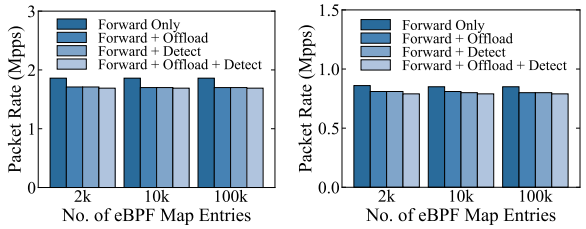
B. Microbenchmark

We first evaluate the impact of flow detection and rules offloading on forwarding performance with a physical/virtual core under the fixed number of entries, as shown in Fig. 8. Here a physical core refers to a hyper-thread on the physical machine, and a virtual core refers to a vCPU in the KVM-based gateway. That is, a physical core and a virtual core occupy the same resources. We want to measure the performance differences of XDP between physical and virtual machines, respectively. We use iPerf [54] to generate UDP traffic, and the inner packet size is 64 bytes. In addition, the number of entries stored in eBPF map ranges from 2k to 100k. As shown in Fig. 8(a), a single physical core can forward 1.86M packets per second under 2k entries. When the rule offloading or flow detection is supplied, the forwarding rate reduces by 8.1% to 1.71Mpps, as these two functions introduce one more eBPF map read/write for flow statistics. After adding both flow detection and offloading functions on the gateway, the performance decreases slightly. For example, the forwarding rate only reduces by 1.2% from 1.71Mpps to 1.69Mpps under 2k entries. This is because the map read/write is the majority overhead for forwarding, while the detection and offloading functions share one eBPF map with only one map read/write.



(a) A physical core (b) A virtual core

Fig. 8. Packet rate under unoptimized timeout mechanism with multiple additional map read/writes vs. no. of entries.

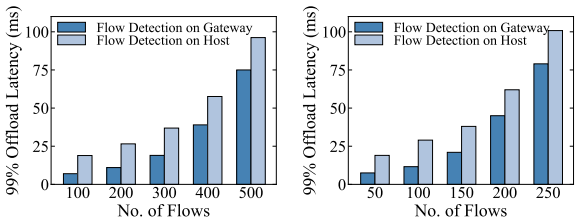


(a) A physical core (b) A virtual core

Fig. 9. Packet rate under optimized timeout mechanism with one additional map read/write vs. no. of entries.

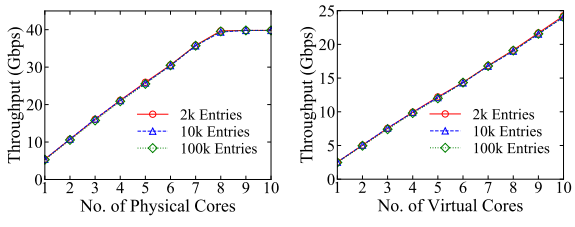
We further evaluate the improvement of map timeout optimization on forwarding performance under an increasing number of entries, as shown in Figs. 8-9. We observe from Fig. 8 that the timeout mechanism with multiple extra map read/writes for flow statistics leads to throughput degradation with the number of entries increasing. We observe from Fig. 8(a) that the forwarding rate with rule offloading and flow detection drops by 14% to 1.45Mpps when the number of entries is 100k. That is because the unoptimized timeout mechanism incurs multiple extra map read/writes, leading to throughput degradation as the number of entries increases. Fig. 9 shows that the optimized timeout mechanism with one additional map read/write can effectively reduce the impact of the number of entries on the forwarding performance. There is no significant performance decline when the map entries scale to 100k, as the read/write overhead of map timeout is nearly O(1) after optimization.

In addition, Fig. 9(a) shows that the pure forwarding rate of one virtual core is 0.86Mpps under 2k entries, which drops



(a) A physical core (b) A virtual core

Fig. 10. 99% Offloading latency vs. no. of flows.



(a) A physical core (b) A virtual core

Fig. 11. Packet rate of a CPU core vs. no. of entries.

54% compared with one physical core. That is because the driver of Intel XL710 VF (*i.e.*, iavf) does not support XDP Native mode in VMs. In order to evaluate the performance of Zeta gateway cluster with a limited number of physical machines, we adopt XDP Generic mode in KVM-based gateways in the following experiments [25], [42]. Considering that the forwarding implement of a single gateway in the comparison scheme (*i.e.*, GWZone) is consistent with Zeta, XDP General mode will not affect the superiority of Zeta in robustness and low forwarding latency. In addition, with the rapid development of XDP, more and more NIC drivers have been supporting the XDP Native mode in VMs (*e.g.*, Intel 82599 and Mellanox ConnectX-5 [55]).

We then measure the rules offloading latency with flow detection on gateway and host, as shown in Fig. 10. The gateway still performs traffic forwarding and rules offloading with a physical/virtual core. We use iPerf to generate UDP flows on a host, each of which is 10Mbps. Fig. 10(a) shows that flow detection on gateway can reduce the 99th percentile of offloading latency by 22% under 500 flows compared with that on host, as the performance of on-host detection is worse than XDP on gateways. In general, flow detection on gateway can reduce the rules offloading latency (*e.g.*, reduce 22% as shown in Fig. 9(a)) and has little impact on the forwarding performance (*e.g.*, decrease 1.2% from 1.71Mpps to 1.69Mpps as shown in Fig. 8(a)). Thus, Zeta detects elephant flow on gateways for faster rules offloading with little detection overhead.

Fig. 11 shows that the total throughput will scale linearly with the increasing number of physical/virtual cores. Specifically, when the inner packet size is 512 bytes and the number of entries is 2k, the throughput of a physical core is 5.4Gbps, and 8 physical cores will hit the NIC's bandwidth limit of the physical machine at 40Gbps. The throughput of a virtual core is 2.54Gbps, and 10 virtual cores will nearly reach 25Gbps throughput. Moreover, with the optimized map timeout mechanism, the throughput does not drop when map entries scale to 100k. From the above results, we can conclude that the linear scaling throughput of CPU cores greatly enhances the scalability of Zeta gateway clusters.

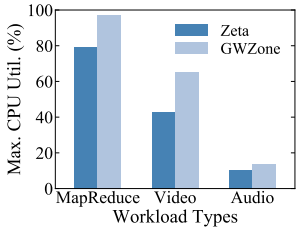


Fig. 12. Max. gateway CPU utilization vs. workload types.

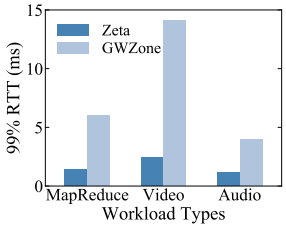


Fig. 13. 99% RTT vs. workload types.

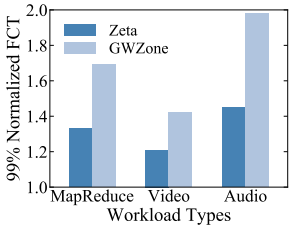


Fig. 14. 99% normalized FCT vs. workload types.

C. Robustness Evaluation

In this section, we evaluate the performance of Zeta under various burst traffic workloads and different abnormal events. According to the trace data from Google cluster-data [53], we deploy 100 VPCs with 2,000 VMs on the 20 compute nodes. Each VPC contains 10-90 VMs, and each VM is equipped with 1 vCPU and 6GB memory.

1) Robustness Under Burst Traffic: We compare the robustness of the Zeta gateway cluster with GWZone under burst traffic of different applications. We choose three typical traffic workloads according to the traffic characteristics in cloud networks [56], [57], including MapReduce, video and audio. Specifically, we deploy a MapReduce cluster in each VPC and execute the word-counting application on each MapReduce cluster simultaneously with input size of 10GB, which mainly generates TCP elephant flows. We also deploy video and audio applications in each VPC. The video traffic contains UDP elephant flows with bandwidth ranging from 2.4Mbps (720P video) to 100Mbps (8K video) [58], [59]. The audio traffic consists of UDP mice flows whose transmission rate ranges from 12.2kbps to 23.85kbps [60].

Figs. 12-15 illustrate the performance metrics of Zeta gateways under different burst traffic scenarios. According to Fig. 10, it takes about tens of milliseconds for gateways to offload the elephant flows, which means that the elephant flows of MapReduce and video applications still bring load to gateways before offloading. Considering that Zeta assigns

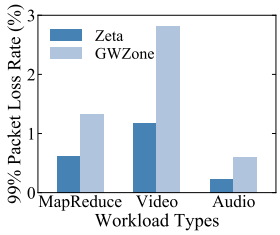


Fig. 15. 99% packet loss rate vs. workload types.

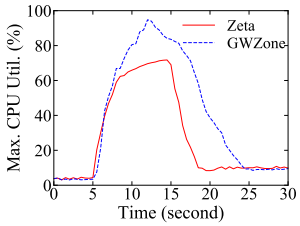


Fig. 16. Max. gateway CPU utilization over time.

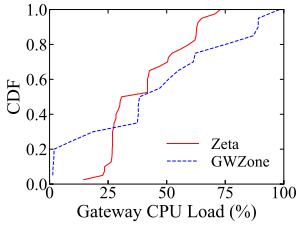


Fig. 17. CDF of gateway CPU utilization.

a gateway cluster to each VPC, while GWZone assigns a primary gateway to each host zone. Thus, Zeta can achieve better load balance to deal with various burst traffic. For example, Fig. 12 records the maximum gateway load during the running of the three applications. Zeta can reduce the maximum gateway load by 18.5%, 33.9% and 25.2% compared with GWZone in the three applications, respectively. In addition, it is noteworthy that the acknowledgment and retransmission mechanisms of MapReduce's TCP flows bring higher load to the gateways than UDP before offloading, which leads to the highest gateway load compared with video and audio streams.

Moreover, Fig. 14 shows the 99th percentile of normalized FCT, which is normalized to the FCT without burst traffic. The 99% normalized FCT achieved by Zeta is 21.3%, 14.8% and 26.8% lower than that of GWZone under three scenarios, respectively. Although the gateway load of audio traffic is low, it mainly consists of mice flows, which will be forwarded by gateways without offloading direct path rules. Thus, the cumulative delay of the audio flows caused by gateway forwarding will be the largest among the three applications, which results in the maximum FCT of audio flows. Besides, we observe from Fig. 15 that the 99th percentile of packet loss rate of Zeta under the three scenarios reduces by 53.8%, 58.2% and 63.3% compared with GWZone. The above results prove that Zeta can effectively conquer different burst traffic and avoid gateways overhead.

Furthermore, we evaluate several performance metrics of Zeta under burst video traffic compared with other frameworks, as shown in Figs. 16-21. During an interval of 0.5s, we record the CPU utilization of gateways, number

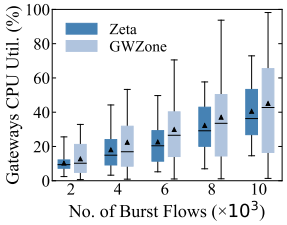


Fig. 18. Gateway CPU utilization vs. no. of burst flows.

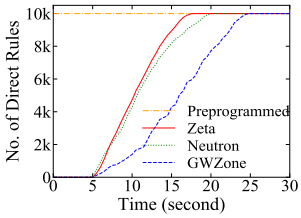


Fig. 19. No. of offloaded direct rules over time.

of offloaded rules, rule offloading latency and FCT. Specifically, Fig. 16 shows the maximum gateway load of Zeta and GWZone in 10k burst video flows. According to the experimental settings, burst traffic are generated randomly in 5-15s, so the average gateways load increases sharply at the 5th second. Next, Zeta detects elephant flows faster on the gateways, so it quickly achieves the balance between offloading and newly coming elephant flows. However, the mice flows continue to increase, so the load of Zeta between 7-15s increases slightly on the basis of stability. Meanwhile, the on-host flow detection of GWZone suffers from high latency, and the elephant flows can not be offloaded in time. Thus, the loads of GWZone's gateways increase sharply from 5s to 13s.

To further study how the workload of gateways distributes, we show the gateways' CPU utilization at the 12th second, when Zeta and GWZone both suffer high gateway workload, in Fig. 18. Zeta achieves lower average load with more concentrated load distribution than GWZone, which means better load balancing. Fig. 17 shows the load CDF of gateways in 10k burst video flows. GWZone's backup gateways are lightly loaded, while 25% primary gateways are overloaded (*i.e.*, the CPU load exceeds 80%). The above results show the superiority of Zeta gateway cluster in load balancing.

Fig. 19 shows the number of offloaded direct forwarding rules in 10k burst video flows. Due to the latency of the on-host flow detection program, the number of offloaded rules for GWZone increases slowly. The number of Zeta offloading rules is increasing rapidly. Preprogrammed is constant at a high point as its preprogrammed model. The trend of Neutron is similar to Zeta. Fig. 21 shows CDF of Normalized FCT in 10k burst video flows. The results are similar to Fig. 19. The preprogrammed model performs the best, followed by Zeta and Neutron, while GWZone is the worst.

2) Fast Recovery From Abnormal Events: We measure the recovery latency of Zeta under abnormal events. Zeta adopts *Multi IPs Migration* for fast recovery, while GWZone updates the default OVS entries on hosts.

Zeta and GWZone both periodically send heartbeat packets to the controller to report the health/load status. Specifically, the gateway sends a heartbeat packet to the controller every

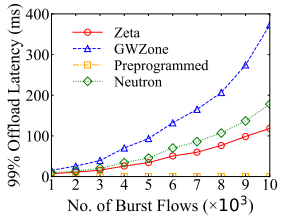


Fig. 20. 99% offload latency vs. no. of burst flows.

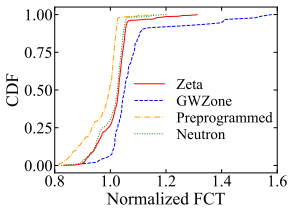


Fig. 21. CDF of normalized FCT.

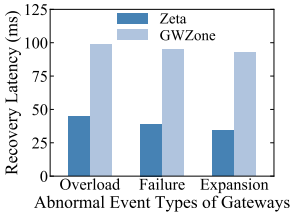


Fig. 22. Recovery latency vs. abnormal events.

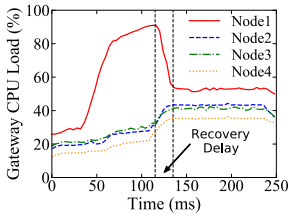


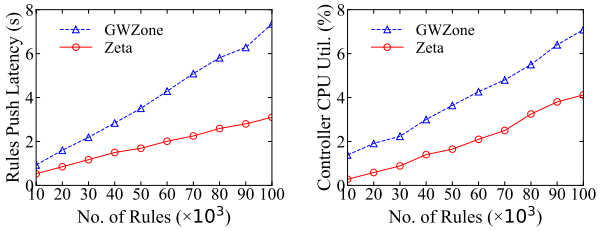
Fig. 23. CPU load of gateways in a cluster over time.

50ms. If the controller does not receive the new heartbeat message from a gateway for more than 55ms, it will actively ask the gateway for the status. If the controller still does not receive the reply from this gateway within the timeout period of 5ms, the gateway is judged to have failed, and the Multi IPs Migration will be performed.

When the measurement starts, we first sequentially send Ping probe packet to a gateway every 0.5ms, and then we make an artificial abnormal event on that gateway. By counting the number of lost packets during the failure recovery, we can derive the recovery delay.

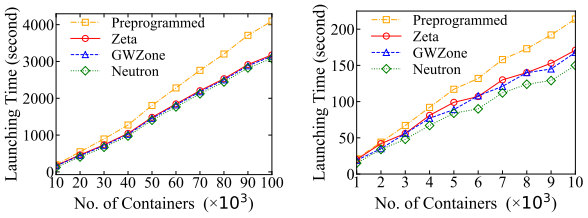
From the results in Fig. 22, we observe that Zeta can effectively reduce the recovery latency of the three abnormal events compared with GWZone. For example, the total failure recovery delay of Zeta gateway is 38.5ms, which is 1.5× faster than that of GWZone, because GWZone needs to inform each host and update ∼100 default entries on each OVS. In addition, if we ignore the failure detection time, the pure recovery delay of Zeta is 5.5 ms, which is 10.8× faster than that of GWZone.

Fig. 23 illustrates the load status of each gateway in a cluster of Zeta during the overload event. Specifically, the burst flows with default destination of Node1 arrive in 35ms, and the CPU load of Node1 increases rapidly. When the gateway's CPU



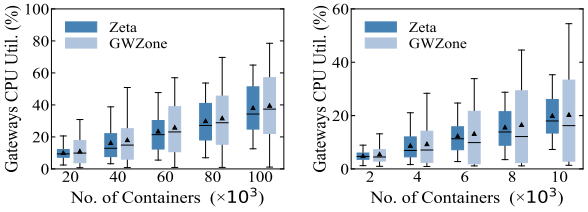
(a) Rules Push Latency (b) Controller CPU Util.

Fig. 24. Controller latency and overhead vs. no. of rules.



(a) The public cloud (b) The private cloud

Fig. 25. Launching time vs. no. of containers.



(a) The public cloud (b) The private cloud

Fig. 26. Gateway CPU utilization vs. no. of containers.

utilization reaches the 90% threshold, the *Multi IPs Migration* is triggered in 120ms, and three vIPs on Node1 are reassigned to the other three nodes with lighter load. Then, the load of Node1 quickly decreases to a normal level within 19.5ms. It is intuitive that *Multi IPs* can effectively conquer the overload of a single gateway and rapidly adjust the load imbalance of intra-cluster.

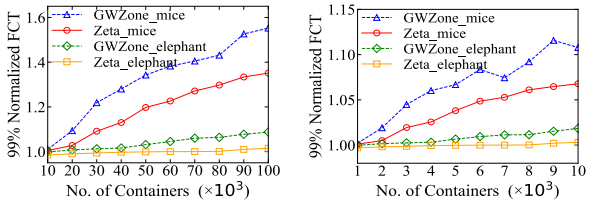
D. Scalability Evaluation

In this section, we evaluate the scalability of Zeta in both public and private cloud scenarios. We first evaluate the latency and overhead of the controller pushing up to 100k forwarding rules. We then measure the latency of launching up to 100k container instances. Finally, we evaluate the performance metrics of Zeta and GWZone under the large-scale cloud network.

The public cloud scenario contains a large number of instances/VPCs. Based on the real trace from Google cluster-data [53], we deploy 568 tenants and 1885 VPCs with up to 100k containers on the 20 compute nodes. Each VPC contains 2-364 containers. The private cloud scenarios have a small number of VPCs/tenants, but a VPC contains a large number of instances. We deploy 52 tenants and 90 VPCs with up to 10k containers on the 20 compute nodes, and each VPC has a number of instances ranging from 2 to 2765.

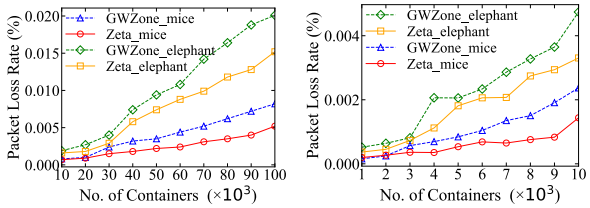
According to the bandwidth distribution of flows in [4], we let 16% of container pairs communicate, and the traffic intensity of each flow ranges from 10kbps to 1Gbps.

1) Rules Push Latency and Overhead: When the forwarding rules initialize (*e.g.*, instances launching) or change



(a) The public cloud (b) The private cloud

Fig. 27. 99% normalized FCT vs. no. of containers.



(a) The public cloud (b) The private cloud

Fig. 28. Packet loss rate vs. no. of containers.

(*e.g.*, instances migration), the cluster controller will push the latest rules to each gateway within the corresponding cluster (in §II-B and §IV-A). Therefore, there is no rule/state synchronization among gateways. In addition, the controller will aggregate multiple rules sent to a gateway into one message, and push rules to multiple gateways in parallel through multiple processes, which effectively reduces the latency and overhead of rules push by the controller.

Fig. 24 shows the latency and CPU utilization of the controller pushing 10k-100k forwarding rules in Zeta and GWZone, respectively. Specifically, both Zeta and GWzone have 45 gateways, of which Zeta's gateways are divided into 10 gateway clusters. We observe from the experimental results that the rules push latency and overhead increase with the number of rules. In addition, Zeta's rules push latency and overhead are lower than GWZone. For example, Zeta controller spends 3.3s pushing 100k rules to 10 gateway clusters, and the push delay is 57.5% lower than that of GWZone. The reason is that the forwarding rule of each instance in Zeta will only be stored in the gateways of one cluster. However, the forwarding rules of GWZone need to be stored in all gateways, because the gateway of the host zone of the source instance requires the forwarding rule of the destination instance, while the source instance may be located in any host zone. Therefore, when a instance boots up or migrates, the controller of GWZone needs to push its forwarding rule to all gateways, leading to higher latency and overhead.

2) Large-Scale Instances Launching: Fig. 25 shows that the on-demand rules offloading model has a lower instance deployment latency compared with the preprogrammed model when spawning a large number of instances in a large-scale cloud network. For example, when launching 100k containers in the public cloud environment, Zeta spends 3178 seconds and installs 12k default forwarding rules, while Preprogrammed spends 4097 seconds and programs a total of 3.4M rules. That is, Zeta reduces the launching time by 24% and the number of rules by 278× compared with Preprogrammed. The reason for the above results is that the on-demand rules offloading can avoid pre-installing numerous entries for instances that never communicate with each other, thus it reduces the latency of instances launching.

3) Large-Scale Instances Communication: Figs. 26-28 show the advantages of Zeta gateway cluster under large-scale networks. As shown in Fig. 26, the average load of Zeta gateways is close to that of GWZone. However, Zeta gateways achieve more concentrated load distribution than GWZone and there is a big gap between maximum and minimum load of GWZone gateways, which means the superiority of Zeta gateway cluster in load balancing.

Next, we evaluate the impact of Zeta and GWZone gateways on FCT. The Normalized FCT of elephant flows and mice flows are calculated respectively. Fig. 27 shows that though Zeta and GWZone have the similar normalized FCT, Zeta still outperforms GWZone by 7% in public cloud scenario, as there is no flow detection load on hosts. In addition, the FCT of elephant flows are both smaller than that of mice flows, because the elephant flows will be forwarded directly.

Finally, we evaluate the packet loss rate of Zeta and GWZone with offloaded elephant flows and non offloaded mice flows to prove the scalability of Zeta. Fig. 28 shows that the packet loss rate of Zeta is lower than that of GWZone because of the better load balancing effect of Zeta gateway cluster. For example, in public cloud with the network scale of 100k containers, the packet loss rate of elephant flows and mice flows of Zeta is 24% and 37% lower than that of GWZone, respectively. In addition, the packet loss rate of elephant flows is higher than that of mice flows. The reason is that these elephant flows will be forwarded by the gateways at the beginning, and burst traffic will cause the gateways overload, resulting in a higher packet loss rate. Therefore, the packet loss of elephant flows is mainly concentrated in the initial gateway forwarding period, and the packet loss of direct path forwarding after offloading will be significantly reduced.

IX. Related Work

Cloud and datacenter virtual networks. There are a multitude of researches on the cloud/datacenter virtual networks, including control plane [61], [62], [63], [64], [65] and data plane [3], [4], [19], [23], [27]. As a crucial solution, overlay network adopts tunnel encapsulation protocols (*e.g.*, VXLAN [36], NVGRE [66], Geneve [67], etc) to build the scalable and flexible virtual networks. Virtual network devices (*e.g.*, vSwitch [32], [68], vRouter [33] and gateway [4]) are essential in the cloud networks, as they are dedicated to provide efficient, secure and stable connections for tenants in clouds. The paper [32] describes the design and implementation of Open vSwitch (OVS), which is an open source and widely used software switch for virtual networks. They focus on the optimization of flow classification and caching to conserve hypervisor resources. The work [33] builds Disaggregated Software-defined Router (DSR) to serve cloud inbound/outbound traffic. They split DSR functionalities into several disjoint modules, each of which can be independently scaled and maintained, to improve the scalability of DSR. The work [4] presents the Hoverboard forwarding model, which adopts gateways to on-demand install forwarding rules and establish communication at scale. In this paper, we focus on

improving the robustness of east-west forwarding with the designs of gateway cluster and multi IPs migration.

High performance and programmable data plane. Data plane is the most performance-critical part of the cloud networks, which is usually accelerated with specialized hardware components and sophisticated software methods [69]. In hardware, ASIC [23], [70], FPGA [19], [20], [21], [22] and network processor [71], [72] can provide high-throughput and low-latency packet processing. For example, the paper [23] proposes Sailfish, a multi-tenant multi-service cloud gateway accelerated by programmable switches, of which a single cluster can carry dozens of Tbps traffic. However, the limitation is that numerous forwarding rules due to multi-tenancy cannot be stored in the limited on-chip memory. In contrast, software methods can provide ample on-host memory and have the advantage of fast and flexible iteration, including DPDK [73], XDP [25], Netmap [74], etc. Unlike XDP, DPDK achieves high throughput through bypassing the kernel network stack and processing packets in user space [73]. However, DPDK suffers from high CPU utilization. Specifically, DPDK needs to bind one or more dedicated CPU cores as it adopts the busy poll mode to pull data packets to the user space, which leads to its CPU utilization always remaining at 100% [38]. In contrast, XDP adopts pre-stack processing mechanism and interrupt mode, which can request CPU resources on demand [25], [38].

eBPF and its applications. Extended Berkeley Packet Filter (eBPF) is an instruction set and an execution environment inside the Linux kernel [38]. It enables injecting custom code into the kernel through various hooks. XDP is one of the most widely used eBPF hooks for high-performance data path that can perform packet processing before the kernel network stack [25]. Currently, eBPF is extensively used in security [75], tracing [76] and networking [77]. In addition, eBPF programs can maintain and access persistent memory through eBPF maps, which are generic key-value stores that serve as the data channel between different eBPF programs or user applications [38], [43]. The work [43] presents BMC to accelerate in-memory key-value store, which adopts XDP for pre-stack requests processing and uses eBPF maps as in-kernel cache. In this paper, we implement the timeout mechanism of eBPF map by ourself, as eBPF map does not support this mechanism. We further optimize the timeout mechanism with one map read/write so that it will not lead to throughput degradation with the increase of entries.

X. Conclusion and Future Work

In this paper, we propose a scalable and robust east-west forwarding framework for hyperscale clouds, called Zeta. Comprehensive experiment results show high robustness and scalability of Zeta. For example, Zeta reduces the 99% RTT by 5.1× in burst video traffic, and reduces the gateway pure recovery delay by 10.8× compared with the state-of-the-art solutions.

With the stagnant performance of CPU core and the rapid growth of cloud scale, the traffic growth will far exceed Moore's law in the future. Although software-based Zeta gateway cluster can realize horizontal scaling, it may not be cost-effective compared with hardware. Fortunately, with the development of programmable hardware, offloading XDP programs to SmartNICs is a promising approach.

Another feasible research direction is to learn the communication pattern between VMs to predict the VM-VM communication and pre-install forwarding rules. Although Zeta improves the robustness and scalability of east-west communication in clouds, the gateways will cause forwarding delay in processing header packets and offloading direct path rules. Therefore, we will adopt machine learning algorithms to obtain the VM-VM communication pattern, so as to pre-install potential forwarding rules to minimize the communication delay.

References

[1] Q. Zhang et al., "Zeta: A scalable and robust east-west communication framework in large-scale clouds," in *Proc. 19th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2022, pp. 1231–1248.

[2] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[3] T. Koponen et al., "Network virtualization in multi-tenant datacenters," in *Proc. 11th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2014, pp. 203–216.

[4] M. Dalton et al., "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 373–387.

[5] Q. Cheng, M. Bahadori, M. Glick, S. Rumley, and K. Bergman, "Recent advances in optical technologies for data centers: A review," *Optica*, vol. 5, no. 11, pp. 1354–1370, 2018.

[6] K.-I. Sato, H. Hasegawa, T. Niwa, and T. Watanabe, "A large-scale wavelength routing optical switch for data center networks," *IEEE Commun. Mag.*, vol. 51, no. 9, pp. 46–52, Sep. 2013.

[7] M. Rzepka, P. Borylo, A. Lason, and A. Szymanski, "PARD: Hybrid proactive and reactive method eliminating flow setup latency in SDN," *J. Netw. Syst. Manag.*, vol. 28, no. 4, pp. 1547–1574, Oct. 2020.

[8] V. Bahl, "Emergence of micro datacenter (cloudlets/edges) for mobile computing," *Microsoft Devices Netw. Summit*, vol. 2015, p. 2-1, 2015.

[9] S. Jha et al., "Measuring congestion in high-performance datacenter interconnects," in *Proc. 17th Usenix Conf. Networked Syst. Design Implement. (NSDI)*, 2020, pp. 37–58.

[10] Google Cloud. (2022). *Containers at Google*. [Online]. Available: https://cloud.google.com/containers

[11] J. Nam, S. Lee, H. Seo, P. Porras, V. Yegneswaran, and S. Shin, "BASTION: A security enforcement network stack for container networks," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 81–95.

[12] S. Choi et al., "FBOSS: Building switch software at scale," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 342–356.

[13] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proc. Conf. Internet Meas. Conf.*, Oct. 2013, pp. 9–22.

[14] J. Wang et al., "A robust service mapping scheme for multi-tenant clouds," *IEEE/ACM Trans. Netw.*, vol. 30, no. 3, pp. 1146–1161, Jun. 2022.

[15] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and mitigating micro-burst traffic in data center networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 98–111, Feb. 2020.

[16] Y. Li, S. J. Park, and J. K. Ousterhout, "MilliSort and MilliQuery: Large-scale data-intensive computing in milliseconds," in *Proc. NSDI*, 2021, pp. 593–611.

[17] Amazon AWS. (2022). *AWS Nitro System*. [Online]. Available: https://aws.amazon.com/ec2/nitro/

[18] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable HPC," *IEEE Micro*, vol. 40, no. 6, pp. 67–73, Nov. 2020.

[19] A. Putnam et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 13–24.

[20] A. M. Caulfield et al., "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.

[21] B. Li et al., "ClickNP: Highly flexible and high performance network processing with reconfigurable hardware," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 1–14.

[22] D. Firestone et al., "Azure accelerated networking: Smartnics in the public cloud," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 51–66.

[23] T. Pan et al., "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 194–206.

[24] M. Oppitz and P. Tomsu, "Software defined virtual networks," in *Inventing the Cloud Century*. Berlin, Germany: Springer, 2018, pp. 149–200.

[25] T. Hoiland-Jorgensen et al., "The express data path: Fast programmable packet processing in the operating system kernel," in *Proc. 14th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2018, pp. 54–66.

[26] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Joint optimization of flow table and group table for default paths in SDNs," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1837–1850, Aug. 2018.

[27] OpenStack Project. (2022). *OpenStack Basic Networking*. [Online]. Available: https://docs.openstack.org/neutron/latest/admin/intro-basic-networking.html

[28] T. A. Bui and M. Canini, "Cloud network performance analysis: An OpenStack case study," M.S. thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2016.

[29] T. Rosado and J. Bernardino, "An overview of OpensTack architecture," in *Proc. 18th Int. Database Eng. Appl. Symp.*, 2014, pp. 366–367.

[30] P. R. Srivastava and S. Saurav, "Networking agent for overlay L2 routing and overlay to underlay external networks L3 routing using OpenFlow and open vSwitch," in *Proc. 17th Asia–Pacific Netw. Operations Manag. Symp. (APNOMS)*, Aug. 2015, pp. 291–296.

[31] A. Saghir and T. Masood, "Performance evaluation of OpenStack networking technologies," in *Proc. Int. Conf. Eng. Emerg. Technol. (ICEET)*, Feb. 2019, pp. 1–6.

[32] B. Pfaff et al., "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2015, pp. 117–130.

[33] H. Shao, X. Wang, Y. Lu, Y. Yu, S. Zheng, and Y. Zhao, "Accessing cloud with disaggregated software-defined router," in *Proc. NSDI*, 2021, pp. 1–14.

[34] K. Poularakis, Q. Qin, L. Ma, S. Kompella, K. K. Leung, and L. Tassiulas, "Learning the optimal synchronization rates in distributed SDN control architectures," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1099–1107.

[35] S. Luo, H. Xing, and K. Li, "Near-optimal multicast tree construction in leaf-spine data center networks," *IEEE Syst. J.*, vol. 14, no. 2, pp. 2581–2584, Jun. 2020.

[36] M. Mahalingam. *Virtual Extensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks*, document RFC7348, pp. 1–22, 2014.

[37] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova, "Overview of GridRPC: A remote procedure call API for grid computing," in *Proc. Int. Workshop Grid Comput.* Cham, Switzerland: Springer, 2002, pp. 274–278.

[38] M. A. M. Vieira et al., "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," *ACM Comput. Surv.*, vol. 53, no. 1, pp. 1–36, Jan. 2021.

[39] I. Marinos, R. N. M. Watson, and M. Handley, "Network stack specialization for performance," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 175–186, Feb. 2015.

[40] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal, "Understanding host network stack overheads," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 65–77.

[41] (2022). *eBPF Maps*. [Online]. Available: https://ebpf.io/what-is-ebpf/#maps

[42] Cilium. (2022). *BPF and XDP Reference Guide*. [Online]. Available: https://docs.cilium.io/en/latest/bpf/

[43] Y. Ghigoff, J. Sopena, K. Lazri, A. Blin, and G. Muller, "BMC: Accelerating memcached using safe in-kernel caching and pre-stack processing," in *Proc. NSDI*, 2021, pp. 487–501.

[44] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "MapTask scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 190–203, Feb. 2016.

[45] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu, "Stateless datacenter load-balancing with beamer," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 125–139.

[46] R. Finlayson, T. Mann, J. Mogul, and M. Theimer. *Reverse Address Resolution Protocol*, document RFC0903, 1984.

[47] Kubernetes Project. (2022). *kubernetes Eviction Policy*. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/eviction-policy/

[48] Kubernetes. (2022). *Kubernetes Scheduler*. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

[49] (2022). *ETCD: A Distributed, Reliable Key-Value Store*. [Online]. Available: https://etcd.io/

[50] E. Mokotoff, "Parallel machine scheduling problems: A survey," *Asia–Pacific J. Oper. Res.*, vol. 18, no. 2, p. 193, 2001.

[51] Gurobi. (2022). *The Fastest Solver*. [Online]. Available: https://www.gurobi.com/

[52] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, and L. Huang, "SAFE-ME: Scalable and flexible middlebox policy enforcement with software defined networking," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–11.

[53] Google. (2020). *Google Cluster-Data*. [Online]. Available: https://github.com/google/cluster-data

[54] iPerf. (2020). *The TCP, UDP and SCTP Network Bandwidth Measurement Tool*. [Online]. Available: https://iperf.fr/

[55] Bootlin. (2020). *The Latest List of NIC Drivers That Support XDP Native Mode*. [Online]. Available: https://elixir.bootlin.com/linux/latest/A/ident/XDP_SETUP_PROG

[56] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, Nov. 2010, pp. 267–280.

[57] J. Lee et al., "Application-driven bandwidth guarantees in datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 467–478, Feb. 2015.

[58] K. Bilal and A. Erbad, "Impact of multiple video representations in live streaming: A cost, bandwidth, and QoE analysis," in *Proc. IEEE Int. Conf. Cloud Eng. (ICE)*, Apr. 2017, pp. 88–94.

[59] C. Canel et al., "Scaling video analytics on constrained edge nodes," 2019, *arXiv:1905.13536*.

[60] L. Kozma-Spytek, P. Tucker, and C. Vogler, "Voice telephony for individuals with hearing loss: The effects of audio bandwidth, bit rate and packet loss," in *Proc. 21st Int. ACM SIGACCESS Conf. Comput. Accessibility*, 2019, pp. 3–15.

[61] T. Koponen et al., "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, vol. 10, 2010, pp. 1–6.

[62] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 254–265.

[63] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Aug. 2012, pp. 19–24.

[64] Y.-W.-E. Sung, X. Tie, S. H. Y. Wong, and H. Zeng, "Robotron: Top-down network management at Facebook scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 426–439.

[65] A. D. Ferguson et al., "Orion: Google's software-defined networking control plane," in *Proc. NSDI*, 2021, pp. 83–98.

[66] M. Sridharan, *NVGRE: Network Virtualization Using Generic Routing Encapsulation*, document RFC 7637, Microsoft, USA, 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7637.html

[67] J. Gross, I. Ganga, and T. Sridhar. *Geneve: Generic Network Virtualization Encapsulation*, document RFC 8926, 2014.

[68] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "Revisiting the open vSwitch dataplane ten years later," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 245–257.

[69] R. Bifulco and G. Retvari, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. IEEE 19th Int. Conf. High Perform. Switching Routing (HPSR)*, Jun. 2018, pp. 1–7.

[70] J. Fang, G. Zhao, H. Xu, C. Wu, and Z. Yu, "GRID: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Trans. Netw.*, early access, Feb. 22, 2023, doi: 10.1109/TNET.2023.3244794.

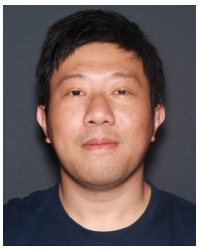
[71] Netronome. (2022). *Agilio CX SmartNICs*. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[72] Nvidia/Mellanox. (2022). *Nvidia BlueField Data Processing Units*. [Online]. Available: https://www.nvidia.com/en-us/networking/products/data-processing-unit/

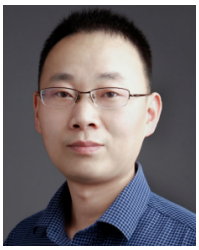
[73] DPDK. (2022). *Data Plane Development Kit*. [Online]. Available: https://www.dpdk.org/
[74] L. Rizzo, "Netmap: A novel framework for fast packet I/O," in *Proc. 21st USENIX Secur. Symp.*, 2012, pp. 101–112.
[75] Falco. (2022). *Cloud Native Runtime Security*. [Online]. Available: https://falco.org/
[76] Bpftrace. (2022). *High-Level Tracing Language for Linux Systems*. [Online]. Available: https://bpftrace.org/
[77] Cilium. (2022). *eBPF-Based Networking, Security, and Observability*. [Online]. Available: https://cilium.io/

Qianyu Zhang is currently pursuing the Ph.D. degree in computer science with the University of Science and Technology of China. His main research interests include software-defined networks and cloud computing.

Gongming Zhao (Member, IEEE) received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is currently an Associate Professor with the University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.



Liguang Xie (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Virginia Tech, Blacksburg, VA, USA, in 2013. He is currently a Senior Principal Architect and the Senior Director of engineering with the Seattle Research Center, Futurewei Technologies Inc., where he leads the Cloud Networking Research and Development Team and oversees cloud networking open-source, research, and engineering efforts. Before Futurewei, he was a Senior Software Engineering Lead with Microsoft Azure Networking and an Adjunct Assistant Professor with the Bradley Department of Electrical and Computer Engineering, Virginia Tech. His research interests include cloud networking, software-defined networking, distributed systems, cloud computing, and AI systems.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China (USTC), China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, USTC. He has published more than 100 papers in famous journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the International Conference on Computer Communications (INFOCOM), and the International Conference on Network Protocols (ICNP). He holds more than 30 patents. His research interests include software-defined networks, edge computing, and the Internet of Things. He received the Outstanding Youth Science Foundation of NSFC in 2018 and the best paper award or the best paper candidate at several famous conferences.

Zhuolong Yu received the bachelor's and master's degrees from the University of Science and Technology of China and the Ph.D. degree from the Department of Computer Science, Johns Hopkins University. His research interests include networking systems, with a focus on programmable networks.

Yangming Zhao received the B.S. degree in communication engineering and the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China in 2008 and 2015, respectively. He is currently a Research Professor with the University of Science and Technology of China. His research interests include network optimization, data center networks, edge computing, and transportation systems.

Chunming Qiao (Fellow, IEEE) is currently a SUNY Distinguished Professor and the current Chair of the Computer Science and Engineering Department, SUNY at Buffalo, Buffalo, NY, USA. He has been working as a consultant for several IT and telecommunications companies since 2000. His research has been funded by a dozen major IT and telecommunications companies, including Cisco and Google, and more than a dozen NSF grants. His current focus is on connected and autonomous vehicles. He has published extensively with an H-index of more than 69. He holds seven U.S. patents. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols. Two of his papers have received the best paper awards from IEEE and Joint ACM/IEEE venues.

Liusheng Huang (Senior Member, IEEE) received the M.S. degree in computer science from the University of Science and Technology of China in 1988. He is currently a Senior Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored six books and over 300 journals/conference papers. His research interests include the Internet of Things, vehicular ad-hoc networks, information security, and distributed computing.

Ying Xiong received the Ph.D. degree in computer and information systems. He is currently the Head of Cloud Lab and the Technical VP with Futurewei Technologies Inc., Santa Clara, CA, USA. He has more than 20 years of experience in distributed system design and large-scale cloud platform implementation. His research interests include cloud and edge computing, virtual networks, and large-scale network management in the cloud, as well as AI platforms for optimizing ML applications.