Zeta: A Scalable and Robust East-West Communication Framework in Large-Scale Clouds

Qianyu Zhang, Gongming Zhao, and Hongli Xu, *University of Science and Technology of China;* Zhuolong Yu, *Johns Hopkins University;* Liguang Xie, *Futurewei Technologies;* Yangming Zhao, *University of Science and Technology of China;* Chunming Qiao, *SUNY at Buffalo;* Ying Xiong, *Futurewei Technologies;* Liusheng Huang, *University of Science and Technology of China*

https://www.usenix.org/conference/nsdi22/presentation/zhang-qianyu

This paper is included in the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Open access to the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Zeta: A Scalable and Robust East-West Communication Framework in Large-Scale Clouds

Qianyu Zhang[1], Gongming Zhao[1*], Hongli Xu[1*], Zhuolong Yu[2], Liguang Xie[3]
Yangming Zhao[1], Chunming Qiao[4], Ying Xiong[3], Liusheng Huang[1]
[1]*University of Science and Technology of China,*
[2]*Johns Hopkins University,* [3]*Futurewei Technologies,* [4]*SUNY at Buffalo*

Abstract

With the broad deployment of distributed applications on clouds, the dominant volume of traffic in cloud networks traverses in an east-west direction, flowing from server to server within a data center. Existing communication solutions are tightly coupled with either the control plane (*e.g.*, pre-programmed model) or the location of compute nodes (*e.g.*, conventional gateway model). The tight coupling makes it challenging to adapt to rapid network expansion, respond to network anomalies (*e.g.*, burst traffic and device failures), and maintain low latency for east-west traffic.

To address this issue, we design Zeta, a scalable and robust east-west communication framework with gateway clusters in large-scale 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 to quickly recover the system/devices 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 speeds up the gateway recovery by 10.8× compared with the state-of-the-art solutions.

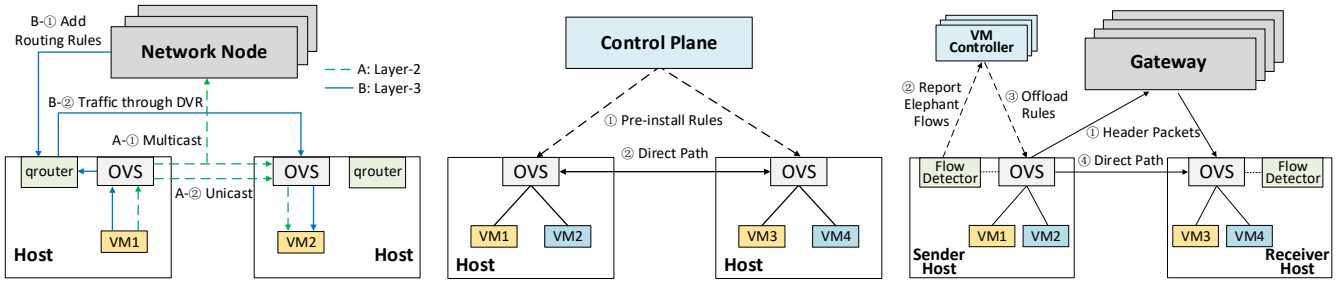
1 Introduction

With an increasing number of distributed applications (*e.g.*, MapReduce [82] and Elasticsearch [32]) on the clouds, east-west communication between instances has become the majority load (even up to 75% [17]) in cloud networks [65]. In addition, cloud providers usually offer isolation for tenants through Virtual Private Cloud (VPC) [77]. Therefore, it is essential for cloud networks to support high-speed and reliable intra-VPC communication [83]. However, two factors bring much pressure on cloud networks. On one hand, a large-scale cloud can accommodate over 100k servers and millions of instances with Pbps bandwidth [7], bringing congestion risks to the network. According to the monitoring log of a cloud with 1,500 servers, we can observe congestions that last over 1s for more than 12,500 times in one day [38]. On the other hand, 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 [31, 50].

As a result, the east-west communication between instances 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 [22], which brings high latency on the rules lookup and traffic forwarding. Therefore, installment of numerous rules will limit the size of a single VPC and the whole network. (2) *Robustness.* Although the failure probability of a specific equipment is usually low, network abnormal events in large-scale clouds are frequent and inevitable, including device failures [18, 59] and burst traffic [68, 81]. They pose severe network congestion/interruption and degrade the tenants' experience. (3) *Latency.* The latency of configuring forwarding rules and establishing/resuming communication is a crucial metric. 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 [47]) 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 communication solutions in cloud networks are usually divided into two main categories. One is the hardware solutions, such as AWS Nitro System [6, 67], Azure FPGA-based SmartNIC [28, 46, 61] and AliCloud P4-based Gateway [57]. The other is the software solutions, including the preprogrammed model (*e.g.*, VMware NSX

*Gongming Zhao and Hongli Xu are the co-corresponding authors.



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

Figure 1: Three Typical East-West Communication 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.

[39, 56]) and the gateway model (*e.g.*, Google Cloud Hoverboard [22]). Considering the high cost and long development cycles of hardware, software solutions have become the preferred choice for many medium-sized cloud providers. However, the existing software solutions also face several critical disadvantages (see §2.1 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 aggravates the recovery delay in network abnormal events, including device failure/overload and VM migration.

To overcome the above challenges, we propose a scalable and robust east-west communication framework in large-scale 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) [36] to accelerate gateway forwarding, thereby enhancing the network scalability and robustness. (2) Zeta adopts the flow table and group table [84] 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 communication models in large-scale clouds and present the design principles for our framework.
- We design a prototype framework, called Zeta, to achieve scalable and robust east-west communication in large-scale clouds. Zeta is publicly available at https://github.com/futurewei-cloud/zeta/.
- We evaluate the robustness and scalability of Zeta through comprehensive experiments. Evaluation results show that Zeta reduces the 99% RTT by 5.1× in burst video traffic, and speeds up the gateway recovery by 10.8× compared with the state-of-the-art solutions.

2 Background and Motivation

We will analyze the limitations of three typical east-west communication models in large-scale clouds and motivate our work in this section.

2.1 Limitations of Prior Works

As an open source cloud computing architecture, OpenStack helps quickly deploy small-scale clouds [63]. As shown in Figure 1(a), OpenStack Neutron provides the networking capability for the clouds. Specifically, Neutron provides layer-2 networking communication by learning MAC address [55]. 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 broadcasts and unicast flooding, leading to poor robustness and scalability [71]. 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 [13], 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 [64]. In general, OpenStack gradually improves forwarding performance through evolutions. But due to the lack of targeted designs for large-scale clouds, it still faces robustness and scalability issues.

Table 1: 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 [13, 55]	✗	✗	✗	✗	✗	✓	✗
Preprogrammed [39]	–	✓	✗	✗	✓	✗	✗
Gateway [22]	✗	✗	✓	✓	✓	✓	✓
Ours: Zeta	✓	✓	✓	✓	✓	✓	✓

To reduce the forwarding latency between VMs, the preprogrammed model was adopted by many early platforms, such as VMware NSX [39, 56]. As shown in Figure 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 nonnegligible system overhead. First, it will pre-install a quadratic number of rules on hosts, which limits the network scalability. Specifically, in a cloud network with h hosts and n VMs, $2n$ rules should be pre-installed before launching a new VM in the worst case, and there will be $O(n \times h)$ rules in the system. 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 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 [22, 39]. Above system overhead leads to poor scalability and flexibility of the preprogrammed model, especially in large-scale cloud networks.

To overcome the disadvantages of the former two models, the gateway model on-demand installs rules, and has been widely adopted by cloud providers, such as Google Cloud Hoverboard [22]. As shown in Figure 1(c), the gateway model organizes all servers into host zones. Host zone/cluster is a collection of colocated machines with uniform network connectivity, each of which is equipped with a master 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 [22], 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 allocates a fixed number of gateway(s) to each host zone and may encounter the robustness issues. 1) *Gateway failure.* Although the master-backup gateway model provides disaster tolerance, it will take a long time to migrate all the traffic from the master gateway to the backup ones, and cannot effectively cope with gateway failures. For example, it takes 260-310ms [72] to inform 14 affected hosts and update the default entries on each OVS. The recovery delay far exceeds the carrier-grade requirements of 50ms [52, 72, 78]. The network interruption caused by the excessive recovery delay will seriously decrease the QoS. 2) *Burst traffic.* The gateway model only assigns a master gateway to each host zone. When a host zone encounters burst traffic, the corresponding master gateway will be easily overloaded (especially when the control plane cannot detect and offload high bandwidth flows immediately).

2.2 Our Intuitions

As summarized in Table 1, 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 master 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.

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 ECMP [69]. 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 [60]. For example, assuming that a large datacenter contains 500 gateways and launches 3k containers per second [31, 50]. The controller needs to send 1.5M update messages in one second, which poses a severe risk of control plane overload. Obviously, this solution is not feasible for large-scale clouds.

In order to integrate the pros, but mitigate the cons of models discussed above, we divide all gateways into multiple clusters. A gateway cluster can effectively improve 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. Moreover, we abstract the gateways'

forwarding capability as Gateway Cluster Layer, which is decoupled from the location and logic of other planes/modules. On the one hand, we utilize gateway clusters independent of the compute nodes to enhance robustness and achieve high performance. We adopt the Multi IPs scheme, which is transparent to hosts/tenants to achieve gateway fast recovery. The independence of gateway cluster gives us flexibility in building high-performance data plane. We choose XDP as the data plane of Zeta, because of its integration with Linux kernel and similar speed as DPDK [24]. On the other hand, the new framework allows easy integration with existing cloud platforms. Thus, we can make full advantage of existing designs, such as Open vSwitch (OVS) [58] group table. According to the above ideas, we design a scalable and robust east-west communication framework in large-scale clouds to support high-performance traffic forwarding.

3 System Design

3.1 Design Goals

Zeta is an east-west communication framework with gateway clusters in large-scale clouds. Our design goals are as follows:

- **Robustness:** High reliability is the core requirement of east-west communication, 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 latency of the traffic forwarding through the high-performance in-kernel fast-path. In addition, the lightweight control loop helps reduce the 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.

3.2 System Overview

As shown in Figure 2, to realize the above design goals, we propose an efficient east-west communication 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 [48]. It leverages XDP to provide high-performance traffic forwarding and on-demand rules offloading for tenant instances (§4.2). The application

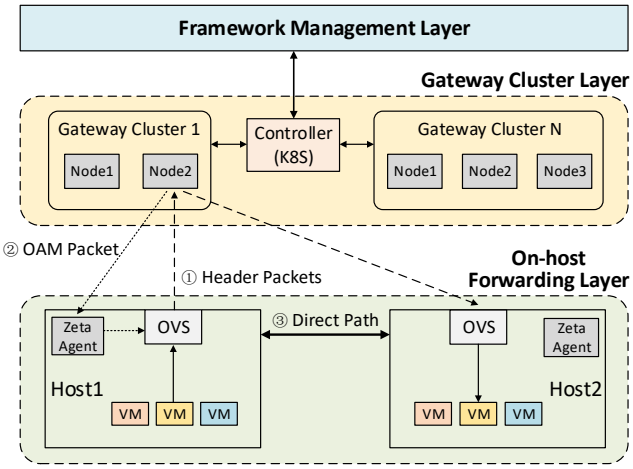


Figure 2: Zeta Framework Overview. Gateway Cluster provides high-performance traffic forwarding and on-demand rules offloading for tenant instances. On-host Forwarding transmits traffic according to default/direct rules and achieves the intra-cluster gateway load balancing through group tables. Framework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

of gateway cluster ensures better scalability and robustness. Gateways detect the elephant flows and sends OAM (Operations, Administration and Maintenance) packets to the source hosts, which contain direct path rules (§4.3). In addition, Zeta adopts *Multi IPs Migration* to achieve fast recovery from gateway failure/overload/expansion, which makes failure recovery transparent to hosts/tenants (§4.4).

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 (§5.1). 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 (§5.2).

Framework Management Layer manages the whole network and further enhances the robustness of gateway clusters. When Zeta is initialized, the management layer will determine the VPC-cluster mapping for inter-cluster load balancing (§6.1). 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 (§6.2).

4 Gateway Cluster Design

4.1 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 Figure 3, Gateway Cluster Layer consists of a cluster controller and several gateway clusters.

Cluster Controller contains management and scheduling logic for gateway clusters. On the one hand, it facilitates the interaction with the Framework Management Layer through its Northbound RESTful API, such as receiving forwarding rules. On the other hand, it manages the gateway clusters and maintains the gateways load balancing through its Southbound API based on gRPC [66]. Cluster Controller is deployed within its own Kubernetes cluster hosted on Zeta control node(s).

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 of the corresponding cluster. In addition, there is no state synchronization among gateways (in §4.3).

4.2 XDP-based Traffic Forwarding

The forwarding module of a Zeta gateway is implemented based on XDP [36] to improve the forwarding performance and reduce the transmission latency. XDP is a high-performance and programmable network data path, which can directly process layer-2 frames at the NIC driver and hence bypass the kernel network stack [12, 36, 79]. As illustrated in the right plot of Figure 3, we converge the forwarding, computing and storage functions together, which eliminates the overhead of network stack processing [14, 49].

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.

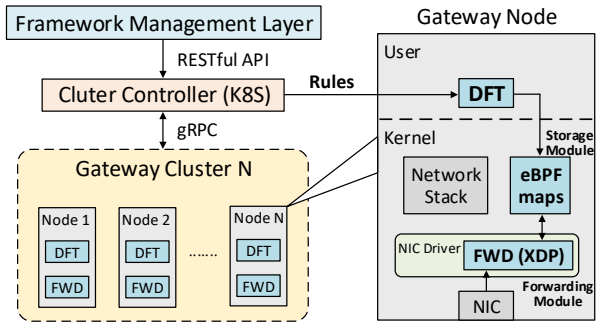


Figure 3: Illustration of Gateway Cluster Design. The left plot is the overview of gateway cluster and the right plot is the implementation details of XDP-based gateway.

Storage Module consists of several eBPF maps [2, 19]. These maps are key-value stores [29] 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 repeatedly offloading one flow.

4.3 Gateway Flow Detection

In order to further reduce the rules stored on the hosts, so as to conserve 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 [82]), 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 [22]. 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 LRU Hash map [44, 79]. If the cumulative size of a flow exceeds the threshold (*e.g.*, 20kbps [22]) 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 §5.1), 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.

4.4 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 §5.1). 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 [54]. 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 [27] to add MAC table entries on the connected 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.

Figure 4 illustrates an example of fast recovery through *Multi IPs Migration*. Initially, Gateway Cluster 1 contains three 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). According to the experiments in §8.3.2, Zeta reduces the average gateway recovery latency from 62ms to 5.5ms.

In conclusion, the *Multi IPs Migration* scheme only needs to update the IPs mapping table and send the RARP packets to ToR switches. The recovery process does not require the participation of control plane or hosts. Therefore, the failure recovery delay and the notification overhead can be almost

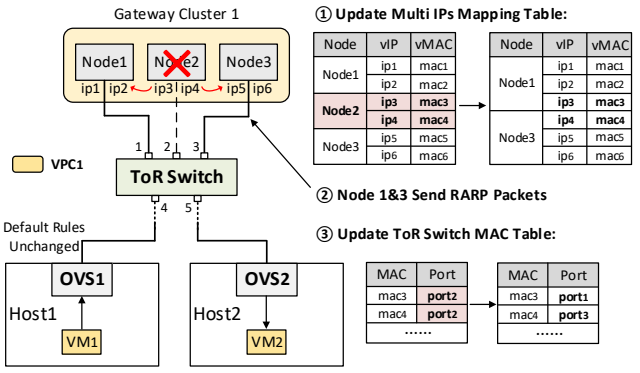


Figure 4: Dealing with Gateway Failures through *Multi IPs*. When Node2 fails, the cluster controller first updates the mapping table to reassign the vIP-vMAC pairs to healthy gateways (*i.e.* Node 1&3). Then Node 1&3 send RARP packets to update the MAC entries on the connected switch. The recovery scheme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

negligible. It significantly enhances the robustness of gateway clusters. In addition, the *Multi IPs Migration* can also be applied in (1) Intra-cluster load adjustment and (2) Rapid cluster scaling (covered in §6.2).

5 On-host Forwarding Design

5.1 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. Specifically, each entry of the group table points to a cluster, and the buckets in each group entry specify the gateway nodes in this cluster. 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 §6.1. Then, the packet will be hashed to a bucket in the group entry, which determines the target gateway for this flow. The group entry selects the target gateway based on the 5-tuple hash of a flow. Finally, the load balancing within a gateway cluster can be guaranteed.

We give an example in Figure 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

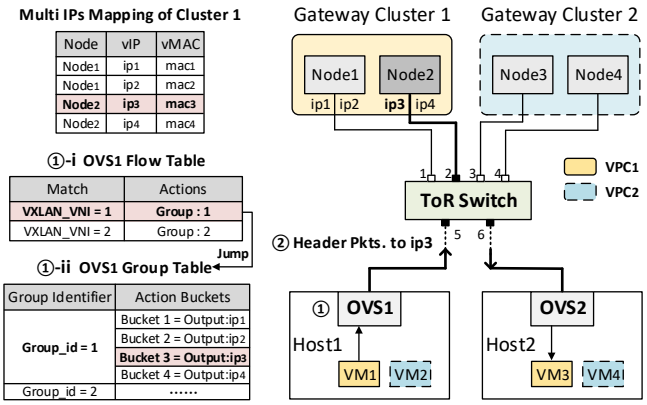


Figure 5: Illustration of Interaction between Flow Table and Group Table. When VM1 belonging to VPC1 communicates with VM3 for the first time, Host1 lookups the OVS1's default tables, and the default gateway IP of VM1's flow is ip2. Then, Host1 sends the header packets of VM1 to Node2.

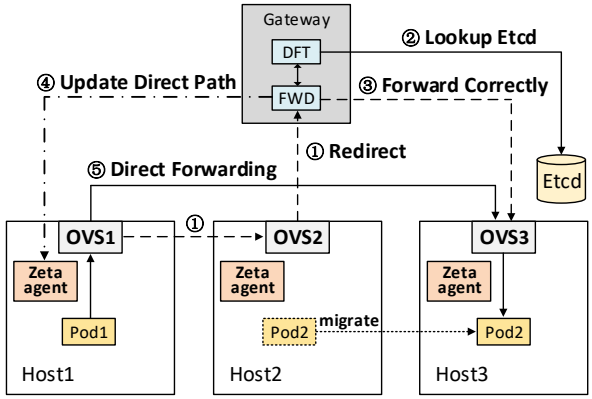


Figure 6: Lightweight Control Agent on compute nodes. When Pod2 is migrated, the flows sent to Pod2 will be redirected to gateway. The gateway forwards the flows and queries the database, then updates the direct path on the source host.

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 information. In our example, the packet is hashed to bucket3, that is, the destination address of the packet is ip3. Then, 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.

5.2 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 [42]. In a Kubernetes cluster, when a compute node is out of resources, the Kubernetes scheduler [43] will migrate the relevant pod(s) to other host(s). Conventionally, Kubernetes does not inform its networking plugin (*e.g.*, Flannel [4] and Calico [1]) of pod(s) migration actively. The networking plugin needs to poll Kubernetes database (*e.g.*, Etcd [3]) to obtain the latest pod information. Therefore, the hosts cannot update the installed direct rules immediately. The traffic is still forwarded to the former destination hosts, which results in a network interruption between the affected pods.

Three steps are required in Zeta to restore communication: (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 *Zeta Agent* remains lightweight to occupy fewer host resources. Meanwhile, Zeta gateways support the above operations. Thus, instead of directly implement above three steps on agent, the traffic towards the migrated pods will be redirected to the gateways and forwarded to the correct destinations.

As illustrated in Figure 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 cache of 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.

6 Framework Management Design

6.1 Gateway Cluster Mapping

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

Gateway Cluster Model. In the Zeta framework, we use $C = \{c_1, c_2, ..., c_n\}$ 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 $V = \{v_1, v_2, ..., v_m\}$ as the VPC set, where $m = |V|$ is the number of VPCs in the cloud. Let $T = \{t_1, t_2, ..., t_{|T|}\}$ denote the tenants set and each tenant $t \in T$ consists of a VPC set $V_t = \{v_1^t, v_2^t, ..., v_{|V_t|}^t\}$. Obviously, $V = V_1 \cup V_2 ... \cup V_{|T|}$. Moreover, the traffic demand of each VPC is denoted as $f(v)$.

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 (§5.1). (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:

$$\min \ \lambda$$

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

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.

We give an empirical formula to set the tenant constraint k in §A.1, and propose a rounding-based algorithm for the VPC-cluster mapping in §A.2.

6.2 Multi IPs Scheduler

The *Multi IPs Scheduler* executes the IPs migration scheme proposed in §4.4. 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* starts to 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.

7 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 provides two deployment methods. One is based on physical machines, and we give a best practice in §B.2. The other is based on Kernel-based VMs (KVMs), which can quickly deploy dozens of KVM-based gateways on several physical machines (see §B.3 for more details).

8 Evaluation

We first conduct an ablation analysis to measure the performance of Zeta gateway. 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.

8.1 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 compute nodes, 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.

Moreover, according to the empirical data in [22], 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 [22], called GWZone, and its gateway is modified based on the implementations of Zeta's gateway. It allocates a master gateway for each host zone and equips backups to deal with gateway failure. 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 master gateways. The second one is the OpenStack Neutron [55], 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 [39, 56] as it is not open source. The Preprogrammed model will pre-install all potential rules before launching VMs.

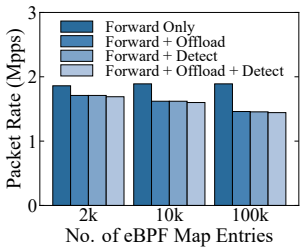


Figure 7: Packet Rate of a Physical Core vs. Entries

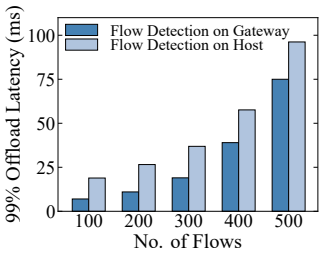


Figure 8: 99% Offloading Latency vs. No. of Flows

8.2 Microbenchmark

We first evaluate the impact of flow detection and rules offloading on forwarding performance with a physical core. We use iPerf [37] 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 Figure 7, 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 additional eBPF map read/writes 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/writes are the majority overhead for forwarding, while the detection and offloading functions require the same number of map read/writes. When the number of entries scales to 100k, the forwarding rate with rule offloading and flow detection drops by 14% to 1.45Mpps, as the timeout mechanism of maps for flow statistics leads to throughput degradation with the number of entries increasing. We will optimize the timeout mechanism in future work.

We then measure the rules offloading latency with flow detection on gateway and host. The gateway still performs traffic forwarding and rules offloading with a physical core. We use iPerf to generate UDP flows on a host, each of which is 10Mbps. Figure 8 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 Figure 8) and has little impact on the forwarding performance (*e.g.*, decrease 1.2% from 1.71Mpps to 1.69Mpps as shown in Figure 7). Thus, Zeta detects elephant flow on gateways for faster rules offloading with little detection overhead.

We also evaluate the linear scaling throughput of Zeta gateways (§C.2).

8.3 Robustness Evaluation

In this section, we evaluate the performance of Zeta under various burst traffic workloads and different abnormal events. Based on the further transformation (§C.1) of Google cluster-data [30], 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.

8.3.1 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 [8, 45], 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) [10, 15]. The audio traffic consists of UDP mice flows whose transmission rate ranges from 12.2kbps to 23.85kbps [41].

Figures 9-12 illustrate the performance metrics of Zeta gateways under different burst traffic scenarios. Zeta assigns a gateway cluster to each VPC, while GWZone assigns a master gateway to each host zone. Thus, Zeta can achieve better load balance to deal with various burst traffic. For example, Figure 9 shows that 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 mechanism of MapReduce's TCP flows further increase the gateway load, which leads to the highest gateway load compared with video and audio streams. Moreover, Figure 11 shows the 99th percentile of normalized FCT, which is normalized to the FCT

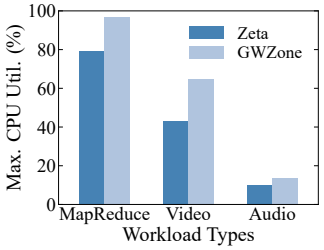


Figure 9: Max. Gateway CPU Utilization vs. Workload Types

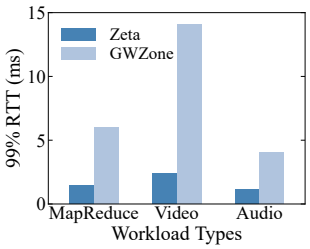


Figure 10: 99% RTT vs. Workload Types

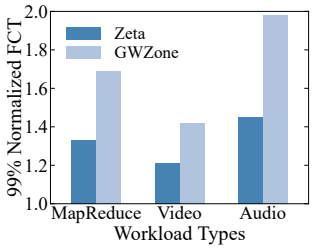


Figure 11: 99% Normalized FCT vs. Workload Types

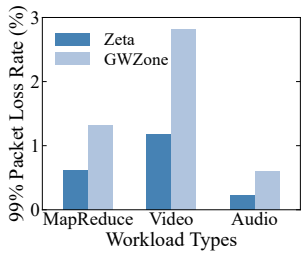


Figure 12: 99% Packet Loss Rate vs. Workload Types

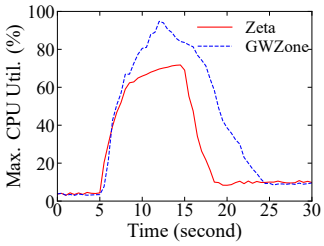


Figure 13: Max. Gateway CPU Utilization over Time

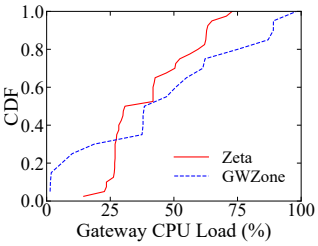


Figure 14: CDF of Gateway CPU Utilization

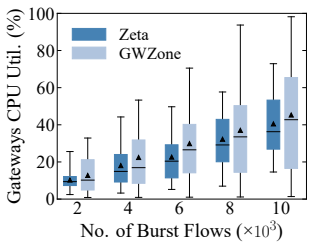


Figure 15: Gateway CPU Utilization vs. No. of Burst Flows

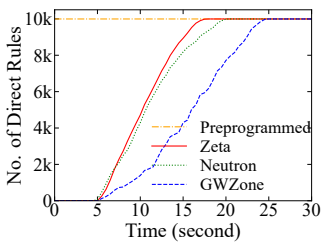


Figure 16: No. of Offloaded Direct Rules over Time

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 Figure 12 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 in burst video traffic compared with other frameworks, as shown in Figures 13-18. During an interval of 0.5s, we record the CPU utilization of gateways, number of offloaded rules, rule offloading latency and FCT. Specifically, Figure 13 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 gateway 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 Figure 15. Zeta achieves lower average load with more concentrated load distribution than GWZone, which means better load balancing. Figure 14 shows the load CDF of gateways in 10k burst video flows. GWZone's backup gateways are lightly loaded, while 25% master gateways are overloaded (*i.e.*, the CPU load exceeds 80%). The above results show the superiority of Zeta gateway cluster in load balancing.

Figure 16 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. Figure 18 shows CDF of Normalized FCT in 10k burst video flows. The results are similar to Figure 16. The preprogrammed model performs the best, followed by Zeta and Neutron, while GWZone is the worst.

8.3.2 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.

Considering that anomaly detection is usually performed by polling, we hope that the delay measurement of failure recovery can avoid the error caused by polling interval. Specifically, we first sequentially send Ping probe every 0.5ms. We

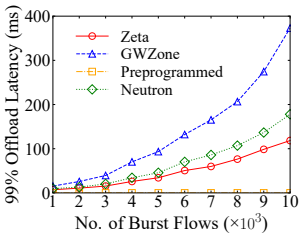


Figure 17: 99% Offload Latency vs. No. of Burst Flows

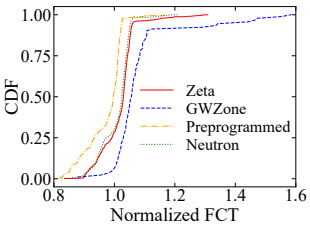


Figure 18: CDF of Normalized FCT

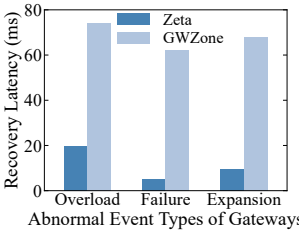


Figure 19: Recovery Latency vs. Abnormal Events

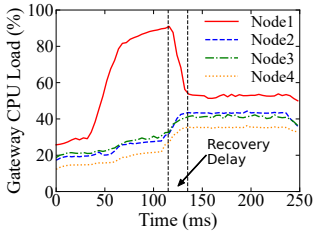
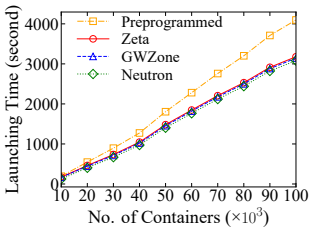
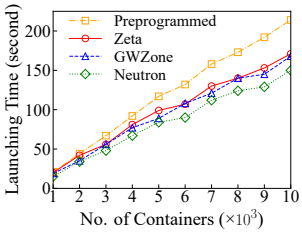


Figure 20: CPU Load of Gateways in a Cluster over Time

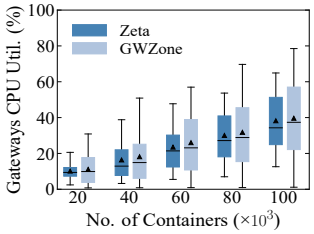


(a) The public cloud

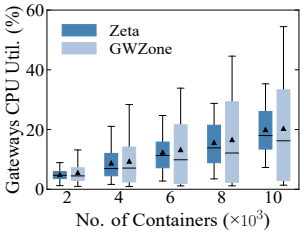


(b) The private cloud

Figure 21: Launching Time vs. No. of Containers



(a) The public cloud



(b) The private cloud

Figure 22: Gateway CPU Utilization vs. No. of Containers

make an artificial abnormal event and notify the controller immediately. Then, the controller performs the IPs Migration. By counting the number of lost packets during the failure recovery, we derive the recovery delay.

From the results in Figure 19, we observe that Zeta can greatly reduce the recovery latency of the three abnormal events compared with GWZone. For example, the gateway failure recovery delay of Zeta is 5.5ms, which is ∼10.8× faster than that of GWZone, because GWZone needs to inform each host and update ∼100 default entries on each OVS.

Figure 20 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 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.

8.4 Scalability Evaluation

In this section, we evaluate the scalability of Zeta in both public and private cloud scenarios. We first measure the latency of launching up to 100k container instances. Then, 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 transformation (§C.1) of Google cluster-data [30], 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 may contain 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.

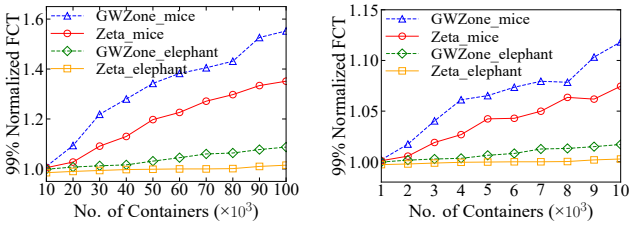
According to the bandwidth distribution of flows in [22], we let 16% of container pairs communicate, and the traffic intensity of each flow ranges from 10kbps to 1Gbps.

8.4.1 Large-Scale Instances Launching

Figure 21 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.

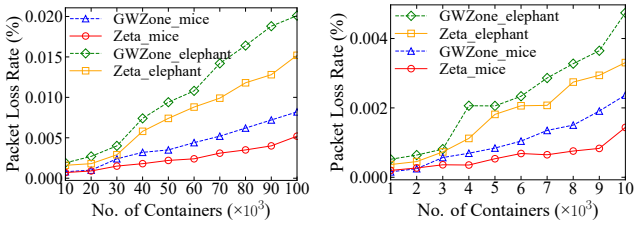
8.4.2 Large-Scale Instances Communication

Figures 22-24 show the advantages of Zeta gateway cluster under large-scale networks. As shown in Figure 22, the average



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

Figure 23: 99% Normalized FCT vs. No. of Containers



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

Figure 24: Packet Loss Rate vs. No. of Containers

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. Figure 23 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. Figure 24 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.

9 Related Work

Cloud and datacenter virtual networks. There are a multitude of researches on the cloud/datacenter virtual networks, including control plane [21, 26, 35, 40, 73] and data plane [22, 39, 55, 61]. As a crucial solution, overlay network adopts tunnel encapsulation protocols (*e.g.*, VXLAN [48], NVGRE [70], Geneve [33], etc) to build the scalable and flexible virtual networks. Virtual network devices (*e.g.*, vSwitch [58, 76], vRouter [69] and gateway [22, 57]) are essential in the cloud networks, as they are dedicated to provide efficient, secure and stable connections for tenants in clouds. 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 [9]. In hardware, ASIC [57, 75], FPGA [16, 28, 46, 61] and network processor [51, 53] can provide high-throughput and low-latency packet processing. In contrast, software methods have the advantage of fast and flexible iteration, including DPDK [24], XDP [36], Netmap [62], etc. Though XDP is not the first mover in this area, we choose XDP as the data plane of Zeta, because of its integration with Linux kernel, interaction with other kernel components and similar speed as DPDK.

eBPF and its applications. eBPF is an instruction set and an execution environment inside the Linux kernel [79]. It enables injecting custom code into the kernel through the hooks. eBPF is extensively used in security [25], tracing [11] and networking [20]. XDP is one of the most widely used eBPF hooks for high-performance packet processing that can bypass the kernel network stack [36].

10 Conclusion and Future Work

In this paper, we propose a scalable and robust east-west communication framework in large-scale clouds, called Zeta. Comprehensive experiment results show high robustness and scalability of Zeta. For example, Zeta speeds up the gateway failure recovery by 10.8× compared with the existing solutions. In future, we will optimize the timeout mechanism of eBPF map to reduce the impact on forwarding performance.

Acknowledgments

We thank our shepherd Minlan Yu and anonymous reviewers for their insightful comments. We also thank the open-source community of Zeta project founded by Futurewei Technologies in 2019 and paper benefits from the original design and implementation of Zeta project. The authors from USTC are supported in part by the National Science Foundation of China under Grant 62102392 and the National Science Foundation of Jiangsu Province under Grant BK20210121.

References

[1] Calico Project, 2022. https://www.tigera.io/project-calico/.

[2] eBPF Maps, 2022. https://ebpf.io/what-is-ebpf/#maps.

[3] Etcd: A distributed, reliable key-value store, 2022. https://etcd.io/.

[4] Flannel Project, 2022. https://github.com/flannel-io/flannel.

[5] David Ahern. XDP and the cloud: Using XDP on hosts and VMs, 2020. https://legacy.netdevconf.info/0x14/pub/slides/24/netdev-0x14-XDP-and-the-cloud.pdf.

[6] Amazon AWS. AWS Nitro System, 2022. https://aws.amazon.com/ec2/nitro/.

[7] Victor Bahl. Emergence of micro datacenter (cloudlets/edges) for mobile computing. *Microsoft Devices & Networking Summit 2015*, 5, 2015.

[8] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280, 2010.

[9] Roberto Bifulco and Gábor Rétvári. A survey on the programmable data plane: Abstractions, architectures, and open problems. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–7. IEEE, 2018.

[10] Kashif Bilal and Aiman Erbad. Impact of multiple video representations in live streaming: A cost, bandwidth, and QoE analysis. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 88–94. IEEE, 2017.

[11] bpftrace. High-level tracing language for Linux systems, 2022. https://bpftrace.org/.

[12] Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. hXDP: Efficient Software Packet Processing on FPGA NICs. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 973–990, 2020.

[13] Tuan Anh Bui and Marco Canini. *Cloud network performance analysis: an OpenStack case study*. PhD thesis, Master's thesis, Université Catholique de Louvain, 2016.

[14] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. Understanding host network stack overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 65–77, 2021.

[15] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536*, 2019.

[16] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[17] Qixiang Cheng, Meisam Bahadori, Madeleine Glick, Sébastien Rumley, and Keren Bergman. Recent advances in optical technologies for data centers: a review. *Optica*, 5(11):1354–1370, 2018.

[18] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. Fboss: building switch software at scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 342–356, 2018.

[19] Cilium. BPF and XDP Reference Guide, 2022. https://docs.cilium.io/en/latest/bpf/.

[20] Cilium. eBPF-based Networking, Security, and Observability, 2022. https://cilium.io/.

[21] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 254–265, 2011.

[22] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, et al. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 373–387, 2018.

[23] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan. High performance network virtualization with SR-IOV. *Journal of Parallel and Distributed Computing*, 72(11):1471–1480, 2012.

[24] DPDK. Data Plane Development Kit, 2022. https://www.dpdk.org/.

[25] Falco. Cloud Native Runtime Security, 2022. https://falco.org/.

[26] Andrew D Ferguson, Steve Gribble, Chi-Yao Hong, Charles Edwin Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, et al. Orion: Google's software-defined networking control plane. In *NSDI*, pages 83–98, 2021.

[27] Ross Finlayson, Timothy Mann, JC Mogul, and Marvin Theimer. RFC0903: Reverse Address Resolution Protocol, 1984.

[28] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[29] Yoann Ghigoff, Julien Sopena, Kahina Lazri, Antoine Blin, and Gilles Muller. BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing. In *NSDI*, pages 487–501, 2021.

[30] Google. Google cluster-data, 2020. https://github.com/google/cluster-data.

[31] Google Cloud. Containers at Google, 2022. https://cloud.google.com/containers.

[32] Clinton Gormley and Zachary Tong. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.", 2015.

[33] Jesse Gross, T Sridhar, P Garg, C Wright, I Ganga, P Agarwal, K Duda, D Dutt, and J Hudson. Geneve: Generic network virtualization encapsulation. *IETF draft*, 2014.

[34] Gurobi. The Fastest Solver, 2022. https://www.gurobi.com/.

[35] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24, 2012.

[36] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies*, pages 54–66, 2018.

[37] iPerf. The TCP, UDP and SCTP network bandwidth measurement tool, 2022. https://iperf.fr/.

[38] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208, 2009.

[39] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, et al. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, 2014.

[40] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[41] Linda Kozma-Spytek, Paula Tucker, and Christian Vogler. Voice telephony for individuals with hearing loss: The effects of audio bandwidth, bit rate and packet loss. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, pages 3–15, 2019.

[42] kubernetes Project. kubernetes Eviction Policy, 2022. https://kubernetes.io/docs/concepts/scheduling-eviction/eviction-policy/.

[43] kubernetes Project. Kubernetes Scheduler, 2022. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/.

[44] Martin KaFai Lau. bpf: Improve LRU map lookup performance, 2017. https://patchwork.ozlabs.org/project/netdev/cover/20170901062713.1842249-1-kafai@fb.com/.

[45] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. Application-driven bandwidth guarantees in datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 467–478, 2014.

[46] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14, 2016.

[47] Yilong Li, Seo Jin Park, and John K Ousterhout. Millisort and milliquery: Large-scale data-intensive computing in milliseconds. In *NSDI*, pages 593–611, 2021.

[48] Mallik Mahalingam, Dinesh G Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. *RFC*, 7348:1–22, 2014.

[49] Ilias Marinos, Robert NM Watson, and Mark Handley. Network stack specialization for performance. *ACM SIGCOMM Computer Communication Review*, 44(4):175–186, 2014.

[50] Jaehyun Nam, Seungsoo Lee, Hyunmin Seo, Phil Porras, Vinod Yegneswaran, and Seungwon Shin. Bastion: A security enforcement network stack for container networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 81–95, 2020.

[51] Netronome. Agilio CX SmartNICs, 2022. https://www.netronome.com/products/agilio-cx/.

[52] B Niven-Jenkins, D Brungard, M Betts, N Sprecher, and S Ueno. Requirements of an MPLS transport profile, 2009.

[53] Nvidia/Mellanox. Nvidia BlueField Data Processing Units, 2022. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[54] Vladimir Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. Stateless datacenter load-balancing with beamer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 125–139, 2018.

[55] OpenStack Project. OpenStack Basic Networking, 2022. https://docs.openstack.org/neutron/latest/admin/intro-basic-networking.html.

[56] Marcus Oppitz and Peter Tomsu. Software defined virtual networks. In *Inventing the Cloud Century*, pages 149–200. Springer, 2018.

[57] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 194–206, 2021.

[58] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, 2015.

[59] Rahul Potharaju and Navendu Jain. Demystifying the dark side of the middle: A field study of middlebox failures in datacenters. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 9–22, 2013.

[60] Konstantinos Poularakis, Qiaofeng Qin, Liang Ma, Sastry Kompella, Kin K Leung, and Leandros Tassiulas. Learning the optimal synchronization rates in distributed SDN control architectures. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1099–1107. IEEE, 2019.

[61] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 13–24. IEEE, 2014.

[62] Luigi Rizzo. Netmap: A novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, 2012.

[63] Tiago Rosado and Jorge Bernardino. An overview of openstack architecture. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 366–367, 2014.

[64] Arsalan Saghir and Tahir Masood. Performance evaluation of openstack networking technologies. In *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–6. IEEE, 2019.

[65] Ken-ichi Sato, Hiroshi Hasegawa, Tomonobu Niwa, and Toshio Watanabe. A large-scale wavelength routing optical switch for data center networks. *IEEE Communications Magazine*, 51(9):46–52, 2013.

[66] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee, and Henri Casanova. Overview of GridRPC: A remote procedure call API for grid computing. In *International Workshop on Grid Computing*, pages 274–278. Springer, 2002.

[67] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A cloud-optimized transport protocol for elastic and scalable hpc. *IEEE Micro*, 40(6):67–73, 2020.

[68] Danfeng Shan, Fengyuan Ren, Peng Cheng, Ran Shu, and Chuanxiong Guo. Observing and mitigating microburst traffic in data center networks. *IEEE/ACM Transactions on Networking*, 28(1):98–111, 2019.

[69] Hua Shao, Xiaoliang Wang, Yuanwei Lu, Yanbo Yu, Shengli Zheng, and Youjian Zhao. Accessing cloud with disaggregated software-defined router. In *NSDI*, pages 1–14, 2021.

[70] Murari Sridharan. Nvgre: Network virtualization using generic routing encapsulation. *draft-sridharan-virtualization-nvgre-00. txt*, 2011.

[71] Piyush Raman Srivastava and Saket Saurav. Networking agent for overlay L2 routing and overlay to underlay external networks L3 routing using OpenFlow and Open vSwitch. In *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 291–296. IEEE, 2015.

[72] Dimitri Staessens, Sachin Sharma, Didier Colle, Mario Pickavet, and Piet Demeester. Software defined networking: Meeting carrier grade requirements. In *2011 18th IEEE workshop on local & metropolitan area networks (LANMAN)*, pages 1–6. IEEE, 2011.

[73] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky HY Wong, and Hongyi Zeng. Robotron: Top-down network management at facebook scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 426–439, 2016.

[74] The kernel development community. Linux TUN/TAP Device, 2022. https://www.kernel.org/doc/html/latest/networking/tuntap.html.

[75] Barefoot Tofino. World's fastest P4-programmable Ethernet switch ASICs, 2018.

[76] William Tu, Yi-Hung Wei, Gianni Antichi, and Ben Pfaff. Revisiting the Open vSwitch dataplane ten years later. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 245–257, 2021.

[77] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A placement vulnerability study in multi-tenant public clouds. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 913–928, 2015.

[78] Luis Velasco. Recovery mechanisms in ASON/GMPLS networks. *Universitat Politècnica de Catalunya (UPC), Barcelona, Spain*, 2009.

[79] Marcos AM Vieira, Matheus S Castanho, Racyus DG Pacífico, Elerson RS Santos, Eduardo PM Câmara Júnior, and Luiz FM Vieira. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)*, 53(1):1–36, 2020.

[80] Jason Wang. Accelerating VM networking through XDP, 2017. https://events19.linuxfoundation.cn/wp-content/uploads/2017/11/Accelerating-VM-Networking-through-XDP_Jason-Wang.pdf.

[81] Jingzhou Wang, Gongming Zhao, Hongli Xu, Yutong Zhai, Qianyu Zhang, He Huang, and Yongqiang Yang. A robust service mapping scheme for multi-tenant clouds. *IEEE/ACM Transactions on Networking*, 2021.

[82] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Transactions On Networking*, 24(1):190–203, 2014.

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

[84] Gongming Zhao, Hongli Xu, Shigang Chen, Liusheng Huang, and Pengzhan Wang. Joint optimization of flow table and group table for default paths in sdns. *IEEE/ACM Transactions on Networking*, 26(4):1837–1850, 2018.

A Additional Details of Cluster Mapping

A.1 Empirical Formula for Tenant Constraint

We use $C = \{c_1, c_2, ..., c_n\}$ to denote the gateway clusters, where $n = |C|$ is the number of clusters. In addition, let I_t denote the number of instance owned by tenant $t \in T$. Then, we use the following empirical formula to set the tenant constraint k:

$$k = \lceil \frac{max_{t \in T}\{I_t\}}{\sum_{t \in T} I_t} \times n \rceil + 1 \quad (2)$$

For example, when our testbed in §8.4 contains 100k container instances, the largest tenant has 27652 instances. We set the tenant constraint $k = 4$, which means the VPCs of a tenant will be mapped to at most 4 gateway clusters.

A.2 Rounding-Based Algorithm

To solve the problem in Eq. (1), we propose a rounding-based gateway cluster mapping (RGCM) algorithm for the GCM problem. The RGCM algorithm includes two steps. The first step is to construct a relaxed version of GCM, named LP-GCM, by relaxing the variable binary constraints. Specifically, LP-GCM assumes that each flow can be splitable and forwarded to multiple gateway clusters. Since LP-GCM is a linear programming, we can derive the fractional solutions $\{\widetilde{x}_v^c\}$ and $\{\widetilde{y}_t^c\}$ with an optimization solver, such as Gurobi [34]. The optimal fractional result is denoted as $\widetilde{\lambda}$.

The second step is to derive the integer solutions with rounding scheme. The integer solutions are denoted as $\{\widehat{x}_v^c\}$

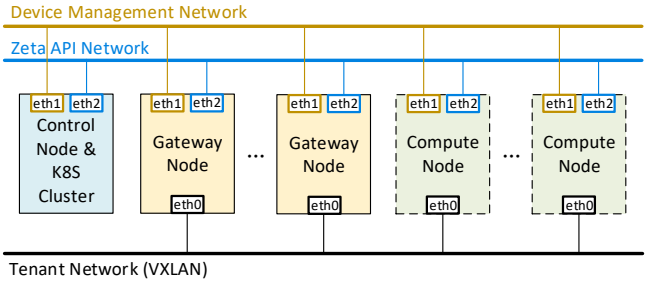


Figure 25: Best Practice for Zeta Physical Deployment.

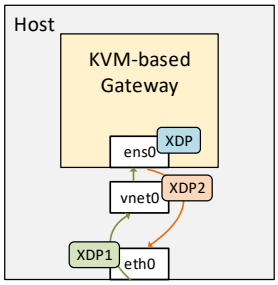


Figure 26: Early Version of KVM-based Gateway.

and $\{\widehat{y}_t^c\}$. For each tenant $t \in T$, RGCM first sorts each gateway cluster $c \in C$ by the value of $\widehat{y}_t^c$ in the descending order. Then RGCM sets the top k maximum $\widehat{y}_t^c$ to 1, which means that the traffic of tenant t can be processed by these k gateway clusters. The set of clusters that are available to the tenant t is denoted as C_t, *i.e.*, $C_t = \{c|\widehat{y}_t^c = 1, c \in C\}$, where $|C_t| = k$. When variables $\{\widehat{y}_t^c\}$ have been determined, RGCM will assign a gateway cluster to each VPC $v \in V$, *i.e.*, determine variables $\{\widehat{x}_v^c\}$. For each VPC $v \in V$, the algorithm selects a cluster $c \in C_t$ with the least burden and sets variable $\widehat{x}_v^c$ to 1.

While solving a linear programming might take a long time for a large network, we note that tenants/VPCs/instances are deployed incrementally, and the number of VPCs/tenants is usually much smaller than that of instances. For example, if hundreds of thousands of instances boot up at the same time, the corresponding VPCs are thousands and the corresponding tenants are hundreds. We utilize Gurobi solver [34] to run the RGCM algorithm on a server equipped with a 10-core Intel i9-10900 CPU. The solution time is 1.15s for the network with 10 gateway clusters, 568 tenants and 1885 VPCs in §8.4, which is acceptable compared to the VPC/instance deployment time.

B Additional Implementation Details

B.1 eBPF Map Size

In the current Linux kernel implementation, the memory usage of an eBPF hash map grows with its entry number. However, the maximum entry size is bounded by the `max_entries` defined by XDP/eBPF program during map initialization. The user space function `bpf_map__resize()` can resize an eBPF map only before it is initialized in the kernel. Unfortunately, we cannot resize an eBPF map after it is created. We have to deploy a new XDP/eBPF program to reinitialize the map size.

Thus, the number of instances that a gateway cluster can serve is limited by the `max_entries` of the eBPF maps. For example, the `endpoint` hash map in Zeta stores instance forwarding rules and its `max_entries` is set to 128*1024 (∼131k). To avoid the above limitation, we can set a larger entry number for the `endpoint` hash map, such as 1024*1024 (∼1M). In addition, the key size and value size of one `endpoint` entry is 8 bytes and 16 bytes, respectively. The total memory size of 1M entries is only 24MB.

B.2 Best Practice for Physical Deployment

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

Figure 25 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 information. The Tenant Network transmits the east-west traffic through the VXLAN tunnel [48] for tenant instances.

B.3 Additional Details of Virtual Deployment

In the early development of Zeta, we use TUN/TAP device [74] as the NICs of KVM-based gateways. In addition to deploying XDP in the KVM-based gateways, we also deploy additional XDP programs on the physical machines to accelerate the host-VM datapath [5,80]. As shown in Figure 26, we attach `XDP1` to the NIC (*i.e.*, `eth0`) of the physical machine to accelerate the host-VM ingress traffic. We attach `XDP2` to the TAP device (*i.e.*, `vnet0`) on the physical machine to accelerate the VM-host egress traffic.

However, Zeta suffers from the poor forwarding performance. For example, the packet forwarding rate of a KVM-based gateway equipped with 4 vCPUs is only 1.36Mpps. The reason is that attaching XDP program to VM's NIC will affect the function of TAP device in host and lead to a significant hit on VM RX performance [5].

Finally, Zeta adopts SR-IOV [23] for KVM-based gateways. Although the driver of Intel XL710 VF (i.e., iavf) does not support XDP Native mode, and Zeta adopts XDP Generic mode with reduced performance in KVM-based gateways [19, 36]. We obtain an acceptable forwarding performance. For example, the pure forwarding rate of one virtual core is 0.86Mpps under 2k entries, which drops 54% compared with one physical core with XDP Native mode.

C Additional Evaluation Details

C.1 Transformation of Google cluster-data

We query the `a.CollectionEvents` table of Google cluster trace and obtain the mapping of `<user,machine,job>` [30]. The machine number is 10001 and the user number is 1952. Considering that we only have 20 compute nodes, while there are 10001 machines in the table. Thus, we merge the jobs of every 500 machines to one compute nodes.

C.2 Linear Scaling Throughput of Gateways

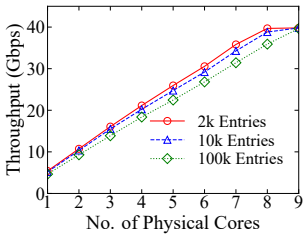


Figure 27: Throughput vs. No. of Physical Cores

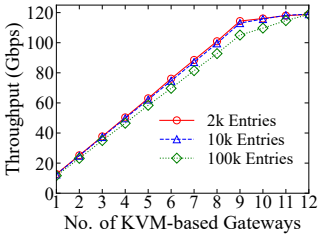


Figure 28: Throughput vs. No. of KVM-based Gateways

Linear Scaling Throughput. Figures 27-28 show that the total throughput will scale linearly with the increasing number of physical cores and KVM-based gateways. Specifically, when the inner packet size is 512 bytes and the number of entries in eBPF maps 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 KVM-based gateway with 4 vCPU is 12.7Gbps, and 9 KVM-based gateways will nearly reach the NICs' total bandwidth limit of the 3 physical gateway machines at 120Gbps. In addition, the timeout mechanism of maps for flow statistics leads to throughput degradation with the number of entries increases. We will try to optimize this issue in future work. The linear scaling throughput of Zeta gateways greatly enhances the scalability of Zeta gateway clusters.