LuoShen: A Hyper-Converged Programmable Gateway for Multi-Tenant Multi-Service Edge Clouds

Tian Pan[†*], Kun Liu[†*], Xionglie Wei[†*], Yisong Qiao[†*], Jun Hu[†], Zhiguo Li[†], Jun Liang[†], Tiesheng Cheng[†], Wenqiang Su[†], Jie Lu[†], Yuke Hong[†], Zhengzhong Wang[†], Zhi Xu[†], Chongjing Dai[†], Peiqiao Wang[†], Xuetao Jia[†], Jianyuan Lu[†], Enge Song[†], Jun Zeng[†], Biao Lyu[‡†], Ennan Zhai[†], Jiao Zhang[◇], Tao Huang[◇], Dennis Cai[†], Shunmin Zhu[⋆†□]

[†]*Alibaba Cloud* [‡]*Zhejiang University* [◇]*Purple Mountain Laboratories* [⋆]*Tsinghua University*

Abstract

Edge clouds are expected to be a key revenue growth driver for cloud vendors in the next decade; however, simply replicating the network infrastructure for the public cloud to the edge experiences deployment issues. At the edge, the challenge for cloud network design is to deliver the required performance under the stringent restrictions of hardware budget and deployment footprints, while retaining functionality equivalence. To this end, we propose *LuoShen*, a hyper-converged gateway for multi-tenant multi-service edge clouds by consolidating the entire cloud network infrastructure into a 2U server switch with a P4-centric architecture. At the data plane, LuoShen conducts pipeline folding and fits the original overlay and underlay devices into the switch pipeline via meticulous on-chip resource budgeting. At the control plane, LuoShen relies on BGP peering to ensure inter-component reachability. LuoShen achieves 1.2Tbps throughput and reduces the upfront cost, deployment size and power usage by 75%, 87%, 60%, compared with the original cloud network architecture. It has been deployed in Alibaba Cloud at hundreds of edge sites.

1 Introduction

The past decade has witnessed the rise of the public cloud to reshape the global IT infrastructure [10]. The success of the public cloud lies in a win-win economic model: cloud vendors save money by bulk purchasing computing resources at lower costs, while cloud customers utilize the shared resources without investing in their own expensive hardware in a pay-as-you-go manner [28, 32, 49]. Following this model, we have built 28 public cloud regions globally for worldwide service coverage [6]. Recently, however, we have seen a surge in customer requests to provide cloud infrastructure close to their sites to address evolving needs, such as ultra-low-latency applications like machine learning inference [56], local data processing of large volumes of data [52], data residency due to security or data sovereignty [43] and the cloudification of telecom infrastructure [23, 29]. These requirements can certainly be addressed by on-premises data centers; however, not everyone can afford to build their own. To meet local processing needs, we start to build edge clouds near our customers with products like Local Region [2] and Cloud Box [1, 3].

To offer customers the same product experience as provided by the public cloud and reduce the time-to-market cycle of our edge clouds, replicating the existing public cloud architecture to the edge is a wise strategy. In our public cloud, to manage traffic bursts from various cloud services and accommodate large forwarding tables due to multi-tenancy, we deploy different roles of gateway clusters for handling different cloud services [41]. However, when extending such "role-splitting" gateway architecture to the edge, we've encountered several deployment issues. The first is how to fit the entire network infrastructure within a constrained space. For example, the Cloud Box condenses the entire cloud infrastructure into a 42U server cabinet. If the network infrastructure consumes too much space, there will be little room left for the server payload, which will reduce the available VMs for sale as well as the vendor's revenue. The second is how to save upfront and operational costs without economies of scale [49]. A typical public cloud region can have tens of thousands of servers, thus the costs of the network infrastructure can be spread over the vast number of servers. However, such economies of scale diminish in a single edge cloud as the network infrastructure constitutes a substantial proportion of its upfront cost due to the much reduced server payload. Moreover, as the number of edge clouds grows rapidly, such cost inefficiency will be magnified many times. The third is how to provide the required stable performance in extreme cases. Although the traffic volumes and the table sizes are significantly reduced at the edge, high-bandwidth traffic and heavy-hitter flows may still occur in some edge cloud use cases, *e.g.*, high-bandwidth traffic directed from the local edge cloud to the remote public cloud, IoT traffic aggregation into a single heavy-hitter flow after being tunneled. Besides, the cloud network infrastructure in production should be durable for a longer service time to save development expenses and adapt to future traffic growth.

[*]These authors contributed equally to this work. [□]Corresponding author.

To address the above issues, we propose *LuoShen*, a hyper-converged programmable gateway for multi-tenant multi-service edge clouds, which *provides the required performance under the stringent restrictions of hardware budget and deployment footprints, while retaining functionality equivalence*. Unlike the expensive role-splitting gateway architecture that uses separate underlay/overlay devices to handle different cloud services like VM-VM (same VPC), VM-VM (different VPC), VM-Cross-region-VM, VM-IDC/IDC-VM, VM-Internet/Internet-VM and SLB, LuoShen fits the entire cloud network infrastructure into a *2U* server switch with a "*P4-centric*" design. That is, all stateless cloud network functions are converged into the Tofino pipeline, with the remaining stateful processing handled by the CPU and accelerated by the FPGA. All traffic will traverse through the Tofino pipeline before being forwarded to the external networks or distributed to the CPU/FPGA by the converged underlay devices. At the data plane, we propose a novel pipeline layout with pipeline folding and carefully budget the on-chip resources to house all the major cloud network functions. At the control plane, we achieve resource isolation, efficient table provisioning and inter-component BGP peering for coexistence of different components. To save the development costs, retain the system stability and shorten the time-to-market cycle, we extensively reuse mature code from the existing public cloud infrastructure. LuoShen has been deployed in Alibaba Cloud for over two years at hundreds of edge sites. We share the experiences and lessons from its development and deployment.

Our major contributions are summarized as follows.

- LuoShen is the world's first hyper-converged gateway disclosed for multi-tenant multi-service edge clouds. It follows a p4-centric architecture and achieves a good balance of performance, costs and deployment footprints.
- At the data plane, we propose techniques such as pipeline folding, pipe/table bypass, on-chip resource budgeting to maximize the table convergence density in the Tofino.
- At the control plane, we reserve multiple configuration channels, and conduct BGP peering with hot standby for inter-component reachability and high availability.
- LuoShen achieves 1.2Tbps throughput and reduces the upfront cost, deployment size and power usage by 75%, 87%, 60%, compared with the role-splitting architecture.

2 Background and Motivation

In this section, we introduce the cloud network infrastructure in Alibaba Cloud over the years, followed by the issues we've encountered when mirroring the infrastructure at edge clouds.

2.1 VPC Network Infrastructure

Virtual private clouds in the public cloud. The public cloud vendors serve a massive number of tenants with a shared infrastructure, where tenant isolation is essential to ensure that their resources are segregated and secure, *e.g.*, one tenant's traffic should be invisible to any other tenant [40]. In addition, some tenants own lots of VMs distributed worldwide but still want to manage them in a unified network address space. To this end, the cloud vendors are expected to virtualize a flat address space for each tenant, hiding the physical network intricacies [26]. To satisfy the isolation and virtualization needs, virtual private clouds (VPCs) [55] are created in the public cloud. The VPC multiplexes the underlying resources and offers a logically isolated address space, and each VPC is uniquely identified by its VNI. Today, overlay protocols, such as VXLAN [37], NVGRE [24], GENEVE [27], are used for VPC implementation. They leverage tunneling to stretch virtualized networks over an underlying L3 network within or across geo-distributed data centers.

Networking requirements of VPCs. A tenant's VMs need to cooperate for delivering scalable cloud services and this produces the VM-to-VM traffic. These VMs can reside in one VPC as the simplest deployment case. They can also reside in multiple VPCs within the same region as some tenants wish to isolate different parts of their infrastructure into different VPCs for more precise access control. A tenant's VPCs can even reside in multiple regions as top tenants deploy VMs across geo-distributed data centers for global service delivery. To this end, the cloud vendors need to address the communication requirements between VMs in the same VPC, in different VPCs within the same region, and across regions. For enterprise customers who use public cloud resources, the connectivity between their on-premises data centers (IDCs) and their VPCs needs to be established. For cloud services through the Internet, to allow Internet traffic to enter into VPCs and vice versa, the connectivity between the Internet and the selected VPCs is also needed. Besides, to handle the massive scale and rapid growth of cloud traffic, either from the Internet (north-south traffic) or from within the data centers (east-west traffic), horizontal scaling of VMs is required and the incoming traffic needs to be directed to the load balancers first, before reaching the backend VMs [18, 39, 42, 57].

Roles of gateways/load balancers in the VPC network. As mentioned, traffic and address space isolation is achieved by assigning one or more VPCs to each tenant, where traffic is isolated within a VPC by default, unless the routes to the external networks are explicitly added to the cloud gateway [31] at the VPC border. Therefore, the first role of a gateway is to route outbound traffic from the local VPC to the external networks and vice versa. To achieve this, the gateway will query its routing table with the VNI of the local VPC and the destination VM IP as the key to obtain the VNI of the next-hop VPC/tunnel if the external network is another virtual network. Besides, traffic will be forwarded across the underlying physical network when being tunneled between two adjacent virtual network devices (*e.g.*, vSwitch [44] or cloud gateway). Therefore, the second role of a gateway is a virtual tunnel endpoint (VTEP), where the outer header encapsulation or decapsulation is conducted when relaying traffic onto or off a physical network domain. When tunneling a packet from the local gateway to a remote VTEP across a physical network,

Table 1: Major cloud services and the corresponding traffic routes in Alibaba Cloud's VPC network infrastructure.

Cloud services	Traffic routes
VM-VM (same VPC)	VM-vSwitch-VGW-vSwitch-VM
VM-VM (different VPCs)	VM-vSwitch-VGW-vSwitch-VM
VM-Cross-region-VM	VM-vSwitch-TGW-Cross-region-TGW-vSwitch-VM
VM-IDC	VM-vSwitch-TGW-CSW-IDC
IDC-VM	IDC-CSW-TGW-vSwitch-VM
VM-Internet	VM-vSwitch-IGW-Internet
Internet-VM	Internet-IGW-vSwitch-VM
Internet-LB-Service	Internet-IGW-SLB-vSwitch-VM
VM-LB-Service	VM-vSwitch-VGW-SLB-vSwitch-VM

the outer SIP will be the gateway's physical IP, the outer DIP will be the remote VTEP's physical IP, and the VNI of the tunnel will also be encapsulated. Furthermore, as most legacy IDCs are non-virtualized (*e.g.*, bare-metal servers) [11, 58], protocol translation is needed for connecting IDCs and VPCs. Similarly, address translation is also required when the gateway provides external connectivity to the Internet. For traffic to be load balanced, it will be delivered to the gateway first, then the gateway will route the traffic to the load balancer. For a better understanding of our design, we elaborate on our gateway's different use cases (see Table 1) as follows.

①VM-VM (same VPC): If two VMs are in the same VPC, inter-VM traffic will first query the VXLAN routing table, confirming that the destination VM is in the local VPC. Then, traffic will be forwarded to the physical server hosting the destination VM with the server IP obtained by querying the VM-NC mapping table using the local VNI and the VM IP.

②VM-VM (different VPCs): If two VMs are in different VPCs within the same region, inter-VM traffic will query the VXLAN routing table twice. In the first pass, we will obtain the VNI of the VPC containing the destination VM. In the second pass, we use the obtained VNI to query the routing table again and the remaining procedure is the same with ①.

③VM-Cross-region-VM: For cross-region VM communication, traffic from the source VM will first query the VXLAN routing table at the local gateway to obtain the VNI of the cross-region tunnel and the physical IP of the remote gateway. After that, traffic will be tunneled to the remote gateway, where its VXLAN routing table will be queried for the VNI of the remote VPC containing the destination VM. The remaining procedure is the same with the second pass of ②.

④VM-IDC: For communication between a virtual network and a non-virtualized network, a specialized device is placed at the border of the public cloud to decapsulate the packets from the virtual network and make the VXLAN-to-VLAN translation if needed. On receiving the outbound traffic from VMs to IDCs, the gateway will first query the VXLAN routing table to tunnel the traffic to the specialized device.

⑤IDC-VM: For inbound traffic from IDCs to the public cloud, the specialized device at the border of the cloud will encapsulate the incoming packets with VXLAN headers and then tunnel them to the gateway of the destination VM. The remaining procedure is the same with ②.

⑥VM-Internet: The VMs in the cloud use private addresses while accessing the Internet needs public addresses. Therefore, the outbound traffic to the Internet needs to query the SNAT table (key: VM IP, value: EIP) at the local gateway to obtain the public address EIP. Then, the gateway will replace the VM IP (*i.e.*, inner SIP) with the EIP and decapsulate the tunnel header before sending the traffic to the Internet.

⑦Internet-VM: The inbound traffic from the Internet will be tunneled to the VM via the gateway. The tunnel encapsulation requires the destination VM IP, the VNI of the VPC containing the VM and the physical address of the server hosting the VM, which can be obtained by querying the DNAT table at the gateway using the EIP (*i.e.*, DIP of the packet).

⑧Internet-LB-Service: Traffic to be load balanced will first hit the gateway before reaching the load balancer. For traffic from the Internet, it will query the DNAT table at the gateway and then be encapsulated and tunneled to the load balancer.

⑨VM-LB-Service: For traffic from the VMs in the cloud (*e.g.*, to use database hosted on servers behind a load balancer), it will query the VXLAN routing table and VM-NC mapping table at the gateway and then be tunneled to the load balancer.

Alibaba Cloud's solution for scalable VPC networking. The challenges of building a scalable VPC network infrastructure lie in many aspects. In a typical cloud region, there are tens of thousands of servers, equipped with 25G/100G NICs, contributing to dozens of Tbps traffic to the gateway. The gateway contains several major tables including the VXLAN routing table, the VM-NC mapping table and the SNAT/DNAT table. In our cloud region, there are O(1M) VPCs and O(1M) VMs, leading to a very large VXLAN routing table, VM-NC mapping table and SNAT/DNAT table. Fitting these tables into an x86 server may not be a serious problem; however, if you want to make hardware acceleration to combat the rapid growth of cloud traffic, the hardware's on-chip memory can easily be exhausted [41]. In Alibaba Cloud, we adopt a *centralized* gateway model that the traffic from the source VM will first be tunneled from its vSwitch to the gateway, where the next-hop forwarding decision will be made. That is, the gateway needs to handle all kinds of traffic as listed in Table 1, leading to a sophisticated packet processing pipeline under ultra-high traffic rates. Moreover, as traffic volumes and cloud services change rapidly, the infrastructure needs to be elastic and flexible. Finally, as the central hub of east-west and north-south traffic, the stability of the gateway is equally important as its performance since failures of the gateway will affect a wide range of tenants and their services. To address these challenges, we have the following architectural designs.

The first design is *separation of underlay and overlay network devices*. Fig. 1 shows the VPC network infrastructure in Alibaba Cloud. There are underlay devices such as SW, LSW, BSW and overlay devices such as vSwitch, SLB and gateways (vendor-specific acronyms are defined in Table A1 in §D). The underlay devices provide underlying network connectivity on top of which overlay tunnels are built. The traffic through the overlay tunnels carried by the underlay devices includes ①east-west traffic between VMs within a region (via

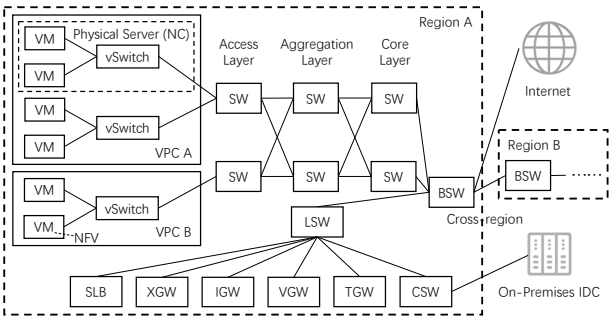


Figure 1: In Alibaba Cloud, different roles of gateway clusters are deployed to forward traffic of different cloud services.

SW); ②traffic routed to the gateways/SLB (via SW, BSW and LSW); ③traffic routed to the external networks (*e.g.*, Internet, via LSW and BSW). The overlay devices provide tenant isolation, virtual routing/forwarding, and tunnel encap/decap. The separation of the underlay and overlay enables rapid cloud service iteration. When cloud services need a change, we only need to update the overlay devices without reconstructing the underlay infrastructure. Besides, caching VM-to-VM routes at vSwitches enables bypassing overlay gateways for faster routing directly through the scalable underlay SW fabric [54].

The second design is *deploying different roles of gateway clusters for different cloud services*. To handle cloud-scale traffic and huge forwarding tables caused by multi-tenancy, the logically centralized gateway is further split into multiple gateway clusters, each dealing with a particular cloud service. For example, *VGW* handles VM-to-VM traffic within a region, *TGW* handles VM-to-VM traffic across regions as well as traffic between VMs and IDCs, *IGW* handles Internet traffic, *SLB* is for server load balancing, *CSW* is a specialized device for VXLAN-to-VLAN translation and vice versa, *XGW* holds all the gateway tables for fallback traffic processing. In the early days, all gateways are based on x86 servers and clustered for scalable performance. To prevent CPU overload due to traffic bursts, some gateways are accelerated via programmable switches [41]. To increase elasticity and flexibility, some stateful gateways, such as NAT and VPN, reside on servers in the form of NFV instances [47]. All the gateways will advertise their VIPs to the underlay network and the vSwitch will tunnel the VM traffic to different gateways by encapsulating the traffic with different outer DIPs based on the VNIs and the inner DIPs. Table 1 shows some major cloud services and their corresponding traffic routes. In the "role-splitting" gateway architecture, failures will be isolated within a single gateway cluster. If horizontal splitting of tenants [41] is enabled, failures will be further isolated within a single physical gateway in the cluster. Similarly, the development, deployment and update of different gateway clusters can also be decoupled.

2.2 Rise of Edge Clouds

Extending public cloud services to the edge. Alibaba Cloud has built 28 public cloud regions across the globe. However, we are increasingly requested by more and more customers to provide cloud infrastructure close to their locations for the following use cases. The first is ultra-low-latency applications such as cloud gaming, live streaming, manufacturing control and machine learning inference [56], which cannot afford the large latencies of task processing in the remote cloud. The second is local data processing needs for large volumes of data generated at the edge, *e.g.*, data uploaded by IoT nodes, which can save network bandwidth usage between the edge and the remote cloud. The third is data residency needs due to security or data sovereignty [43], *e.g.*, financial services. Another potential market is the cloudification of telecom infrastructure, *e.g.*, cloud-native 5G core and RAN [23, 34]. To meet the above needs, we replicate our public cloud infrastructure to the edge, expecting to continue its past success. In this way, there is no need for our customers to build and operate their own on-premises IDCs and they can use the same tools and APIs that they use in the public cloud to manage their edge clouds. Specifically, we have launched two edge cloud products, namely, *Local Region* [2] and *Cloud Box* [1, 3]. The Local Region is a local public cloud infrastructure serving nearby customers in the same city while the Cloud Box is a highly-converged public cloud infrastructure within a 42U server cabinet. They both offer a user experience akin to the public cloud but with significantly reduced capacities, *e.g.*, a Local Region usually has dozens of servers, and a Cloud Box contains even fewer. As a comparison, our largest public cloud region houses hundreds of thousands of servers.

Deployment constraints of edge clouds. When extending the VPC network infrastructure to the edge, we've experienced the following deployment issues. ① *Fitting the entire public cloud into limited space.* Edge clouds have small footprints but full functionality of the public cloud. For example, for Cloud Box, the entire cloud infrastructure (including compute, storage, network, power supply, cooling system) will be packed into a 42U server cabinet, leaving limited space to accommodate servers running tenants' VMs. In addition, if the network occupies too much cabinet space, the server payload will be further compacted, which reduces the available VMs for sale as well as the cloud vendor's revenue. If we pack the 9 different types of underlay and overlay devices (as shown in Fig. 1) with 1:1 backup into the 42U cabinet, there will be little room left for the server payload. ② *Cost disadvantages without economies of scale.* In a public cloud region, the VPC network infrastructure consists of a few clusters of underlay and overlay devices, serving tens of thousands of servers. The upfront and operational costs of the VPC network infrastructure can be spread over the huge number of servers due to the economies of scale [49]. However, for a single edge cloud, the situation has changed as the network infrastructure will occupy a significant proportion of its upfront cost due to the much reduced server payload. Moreover, the "role-splitting" architecture is not conducive to operational cost reduction. Last but not least, as the edge clouds are placed near the customer sites, they will have a huge site number compared with the public clouds. For example, Alibaba Cloud now has only

28 public cloud regions but the number of edge clouds grows rapidly. That is, the cost inefficiency of the original VPC network architecture in a single edge cloud will be magnified thousands of times when edge clouds become more popular.

Possible solutions and their limitations. The traffic load is significantly reduced at the edge due to the fewer tenants served. Therefore, the 3-tier switch fabric for underlay traffic processing can be simplified and the role-splitting gateway clusters for overlay traffic handling are indeed overkill. Accordingly, in our first generation of edge cloud products, we simply use a few ToR switches for the underlay and a few x86 server-based gateways and load balancers for the overlay. Specifically, for the Local Region, we use separate x86 servers to build XGW and SLB. As the XGW holds all the gateway tables, it can easily replace all the single-role gateways, significantly saving the upfront and operational costs as well as the deployment footprints. For the Cloud Box, to maximally save the cabinet space, we further converge the XGW and the SLB into a single x86 server, using different groups of CPU cores to perform different tasks. To handle potential traffic growth in the future, horizontal scaling [41, 50] is still leveraged.

Although extensive horizontal scaling will cause cost inefficiency and large deployment footprints of the VPC network infrastructure, we believe this will not happen soon at the edge. However, two real edge cloud use cases change our minds. The first is about the high-bandwidth requirement at the edge. In this case, our customer expects to migrate his data from the local IDC to the remote public cloud through the nearest point of presence (PoP). However, the nearest PoP is in another city hundreds of miles away from the customer IDC and a leased line from an ISP is needed for the IDC-PoP connection. To save expenses and time, our customer decides to route his data to the public cloud directly through the Local Region in his city, which produces hundreds of Gbps traffic and floods the XGWs in the Local Region. The second is about CPU overload by a heavy-hitter flow. In this case, our customer uses the Cloud Box to manage his IoT devices. The data collected by the IoT devices is sent through a tunnel to the Cloud Box. The tunnel aggregates the traffic into a single flow that reaches dozens of Gbps. At the XGW, the heavy-hitter flow is hashed to a CPU core via the RSS mechanism [25], which easily overloads the CPU core and causes packet drops.

3 LuoShen's Architecture

In this section, we list the design goals of LuoShen, followed by its architectural innovation to achieve these goals.

3.1 Design Goals

①*Small deployment footprints*. The VPC network infrastructure at the edge should have small footprints to leave more room for the server payload, maximizing the VM capacity.

②*Complete VPC network functions*. At the edge, our customers want ultra-low latency but do not want to compromise the consistent product experience of the public cloud. The VPC network inside the 42U server cabinet needs to provide the same functions as provided by the public cloud.

③*Cost efficiency*. Considering the large edge cloud number, the upfront and operational costs of the network infrastructure in each edge cloud should be controlled. Besides, network architectures with smaller energy footprints are preferred.

④*Performance stability*. In a multi-tenant cloud, failures of the shared network infrastructure will affect a large number of tenants. As the gateway is the central hub of the cloud traffic, its performance stability needs to be strengthened to avoid being overloaded by either high-bandwidth traffic or heavy-hitter flows. Once a failure occurs, traffic should be taken over by the backup component as soon as possible.

⑤*Elasticity and flexibility*. Different edge cloud use cases have different network customization needs in terms of traffic scale and network functionality, so the edge cloud network infrastructure should be elastic and programmable to quickly respond to the changing service requirements.

⑥*Avoid reinventing the wheel*. We have invested tremendous person-months into developing the network infrastructure for the public cloud. The stability of the system has stood the test of time. As the edge clouds inherit all the functions of the public cloud, to save the development costs, retain the system stability and shorten the time-to-market cycle, we'd better reuse as much code as possible from the existing systems.

3.2 Hyper-Converged Gateway

Opportunities for infrastructure convergence. Programmable switches [15, 16] have been proved to work well even under cloud-scale traffic bursts [41]. They are suitable for stateless forwarding which covers a majority of cloud gateway functions. However, some cloud services are stateful (*e.g.*, SLB, NAT) and better to be processed by the CPU due to the large memory footprints as well as the high processing complexity. To deal with high-bandwidth traffic, the stateful processing can also be offloaded to the FPGA. In the following, we discuss how to converge the core functions of the VPC network infrastructure into a Tofino chip [5] and how to converge the remaining functions into the CPU with high-performance functions accelerated by the FPGA.

①*Converge different gateway functions sharing the same table*. As you may notice in §2.1, many gateway functions in our cloud share the same forwarding tables, *e.g.*, the VXLAN routing table is involved in all gateway functions except Internet-related services. Hence, we can converge these functions from different gateways into the same Tofino pipeline stages. Actually, in past deployments, we have already made efforts in this direction, *e.g.*, VGW handles VM-VM (same VPC), VM-VM (different VPCs) and VM-LB-Service traffic within a region and TGW handles all the cross-region traffic between two VMs and between VMs and IDCs. Actually, VGW and TGW also share the VXLAN routing table, that is, they can be further converged into a single gateway. Besides, given that Tofino is programmable, we can also converge the VXLAN-to-VLAN translation function into it, saving space originally for CSW. Finally, in the Tofino, we converge VGW, TGW and CSW into a converged gateway called *CGW*.

②*Converge different gateway functions without table overlapping*. In edge clouds, the gateways serve tenants a few miles away, so there is a significant reduction in the number of table entries compared to the public cloud. As a result, it is possible to converge different gateway functions into the same Tofino chip even though their tables have no overlap. Specifically, we can place different functions in different pipeline stages. Packets will be sequentially processed by each function when they traverse through the pipeline. In this way, we can further converge *CGW* and *IGW* into the Tofino chip.

③*Converge underlay and overlay devices*. After we converge overlay devices (*i.e.*, CGW, IGW) into the Tofino, there are still many standalone underlay devices such as SW (for switching east-west traffic), LSW (for routing traffic to gateways) and BSW (for connecting Internet/remote regions), which need additional costs as well as space for placement. Given that Tofino has ultra-high throughput, if its on-chip resources are still available, we can further converge the underlay devices together with the overlay devices into the same gateway. Specifically, we converge the original SW and LSW into a *new* SW, which is further converged with the CGW and IGW into the Tofino (BSW is still standalone). In this way, one Tofino handles both overlay and underlay traffic.

④*Process fallback traffic and stateful forwarding at the CPU*. The P4 switch pipeline cannot handle all types of cloud traffic. Some traffic needs fallback processing. For example, the VM-to-VM route caching at vSwitches requires special treatment of the first packet of the VM-to-VM traffic within a region. If a packet from a VM cannot hit the route cache at the vSwitch, it will be forwarded to the gateway [54]. After querying the VM-NC mapping table at the gateway, a reply packet containing the mapping relationship will be constructed and sent back to the vSwitch for route cache population. However, the reply packet construction is a bit complicated for the P4 pipeline and better to leave to the CPU. Besides, some cloud services (*e.g.*, NAT, VPN) require stateful processing at the gateway with large session tables. These tables are too large to fit into the Tofino and better to use CPU to process them.

⑤*Offload high-bandwidth stateful forwarding to the FPGA*. Although we can use kernel-bypass techniques (*e.g.*, DPDK) [13] to accelerate CPU-based stateful packet processing, for some stateful cloud services, high-bandwidth traffic can easily overload a CPU core [36, 41, 48]. One example is east-west traffic load balancing. Sometimes, a massive number of VMs want to access the cloud service hosted on a server cluster behind a load balancer. The load balancer may face huge traffic pressure since the east-west traffic in a data center is usually not rate-limited. As a comparison, the Internet-related and cross-region traffic are often rate-limited based on tenants' spending. To handle high-bandwidth stateful forwarding, we can use FPGA with large HBM to meet both search speed and memory capacity requirements [57].

The server-switch hardware. According to the above infrastructure convergence principles, we build LuoShen, a hyper-converged programmable gateway for multi-tenant multi-service edge clouds. LuoShen fits the entire VPC network infrastructure into SNA [8], a 2U server switch developed by Alibaba Cloud's infrastructure team, with full public cloud network functionality retained. Inside LuoShen, the original single-role gateways (including VGW, TGW, CSW, IGW) and the underlay devices (including SW and LSW) are converged into a Tofino chip, handling both overlay and underlay traffic; the XGW and SLB are converged into a CPU, handling fallback traffic and conducting traffic load balancing, respectively; the SLB+ in an FPGA conducts traffic load balancing hardware acceleration. LuoShen essentially follows a "P4-centric" architecture inherited from our centralized gateway model in the public cloud (§2.1), because most of the gateway functions are now converged into the P4 switch pipeline. In the public cloud, the cloud gateways are the central hub of the east-west and north-south traffic. Now, for edge clouds, the Tofino in LuoShen becomes the new central hub of the edge cloud traffic. In LuoShen, the 64×100G ports of Tofino are split into different purposes. Some ports directly connect to servers to receive VM traffic, some ports connect to BSW for Internet access and cross-region communication, some ports connect to on-premise IDCs, and some ports connect to the CPU and FPGA for fallback and stateful traffic processing (the CPU sits on a NIC for hardware offloading/traffic rate-limiting, while the FPGA is plugable so that we can also select other cards for fast adapting to the diverse edge cloud service needs). Besides, a large number of ports that are not shown explicitly in Fig. 2 are used internally for pipeline folding [41], which will be detailed in 4.1. As a comparison, in the public cloud, all ports of the P4-accelerated gateways are connected to the upstream switches (*i.e.*, LSW). Since we converge the 8 standalone overlay and underlay devices into a 2U box, LuoShen has superior advantages in terms of deployment footprints, upfront and operational costs, and power consumption for edge cloud deployment.

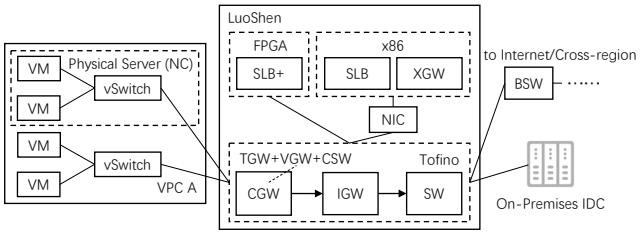


Figure 2: LuoShen fits the entire VPC network infrastructure into a 2U server switch with full functionality retained.

Functionality equivalence analysis. To provide tenants with the same product experience as that of the public cloud, we need to achieve *functionality equivalence* after convergence. We compare the two architectures before and after convergence (Fig. 1 and Fig. 2). First, all the gateways/SLB functionality is retained because VGW, TGW, CSW, IGW are now in the P4 pipeline while XGW and SLB are now in the CPU/FPGA. Second, the functionality of east-west traffic switching as well as traffic routing to

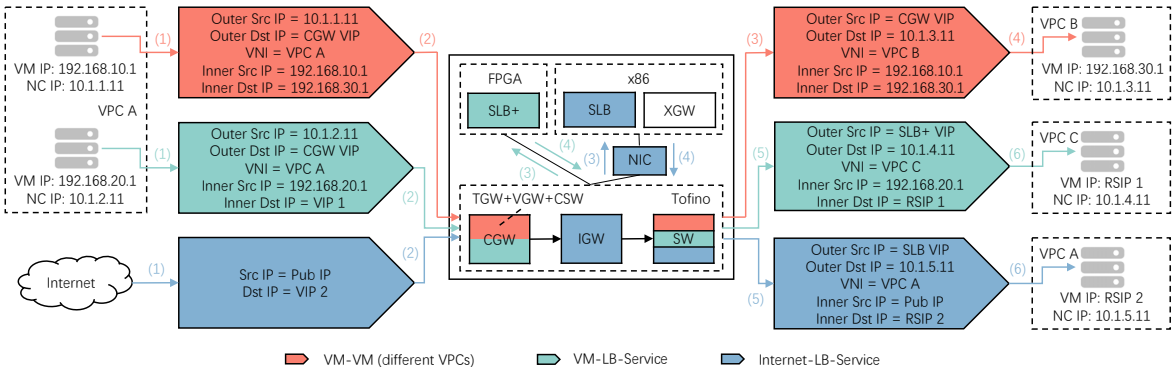


Figure 3: Packet journeys of three different cloud services converge at LuoShen.

gateways/SLB/Internet/IDCs/remote regions provided by the original underlay device SW and LSW is also retained after they are converged into the new SW in LuoShen. Specifically, the original SW needs to route the VM traffic to the gateways/SLB through LSW, however, this function is no longer needed since physical servers are now directly connected with LuoShen. It also needs to switch the east-west traffic which hits the route cache at vSwitches and this function is still retained at the new SW. The original LSW needs to route traffic to gateways/SLB/Internet/IDCs/remote regions. Since VGW, TGW, CSW, IGW are now in the Tofino which is directly connected with physical servers, there is no need to explicitly route traffic to these gateways. However, the function to route traffic to XGW/SLB/Internet/IDCs/remote regions of the original LSW is still retained at the new SW, which will forward the corresponding traffic through the P4 switch ports to these external destinations outside the Tofino.

To summarize, in LuoShen's P4-centric architecture, the core functions of the VPC network infrastructure are offloaded to the P4 pipeline. All incoming traffic will go through CGW, IGW and the new SW sequentially in the pipeline before being distributed to CPU/FPGA or external networks by the SW. Since all traffic experiences the "distribute after deep pipelining" packet flow, the P4 pipeline becomes the traffic aggregation point. Thus, the traffic of different cloud services will be queuing together in the pipeline. However, there is no need to worry about it at present as the high-capacity pipeline is sufficient to accommodate the traffic from the edge.

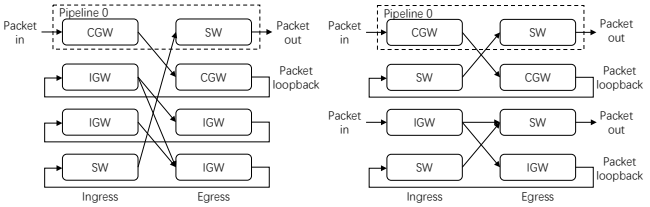
Packet journeys in LuoShen. Fig. 3 shows the packet journeys of three different cloud services converging at LuoShen. The first is VM-to-VM traffic across VPCs within a region, where the outer IP is set to CGW VIP by the vSwitch due to a route cache miss. Then, the packet will be routed to the CGW in LuoShen and query the VXLAN routing table and VM-NC mapping table to obtain the VNI of the destination VPC as well as the server IP of the destination VM. Finally, the packet will query the FIB in SW and be forwarded to the destination VM. The second is VM traffic load balancing for accessing the cloud service behind SLB, which is more complicated as the Tofino needs to interact with the SLB+ in the external FPGA (we use FPGA for SLB acceleration). The packet containing VIP 1 (*i.e.*, the IP of the cloud service) as the inner DIP will be routed to CGW in LuoShen to query the VXLAN routing table and VM-NC mapping table to obtain the VIP of SLB+. After that, the packet will be forwarded by SW to the SLB+, where server load balancing is performed by querying a session table to find the real server (identified by RSIP 1 and 10.1.4.11 in Fig. 3) using the 5-tuple. Then, the packet will be reencapsulated with the real server address as the inner/outer DIP and routed back to the Tofino (because the FPGA has no other connection to the network). Finally, the packet will be forwarded by SW to the selected real server behind SLB+. The third is Internet traffic load balancing for accessing the cloud service behind SLB. The packet flow is almost the same with the second case, except that the traffic from the Internet will be routed to IGW first to query the DNAT table using the DIP to obtain the VIP of SLB, and the server load balancing is performed in the CPU this time.

4 Data Plane

In this section, we show how to fit the core functions of the VPC network infrastructure into a single Tofino chip through sophisticated pipeline layout and on-chip memory budgeting.

4.1 Tofino Pipeline Layout

Pipeline folding for CGW/IGW/SW convergence. The Tofino has 4 pipelines and each pipeline consists of an ingress pipe and an egress pipe, connected by a traffic manager (TM). Each pipeline has limited SRAM/TCAM memory resources [30], distributed equally in 12 stages of that pipeline. Each pipeline stage is shared by the ingress pipe and the egress pipe, *e.g.*, if we fit a large table into stage 0 of the ingress pipe, there will be little space left for stage 0 of the egress pipe. When converging the core functions of the VPC network into the Tofino, we should first evaluate if the stage/memory resources are sufficient to hold the deep pipeline of CGW, IGW, and SW. Actually, in our P4-based single-role gateways for the public cloud [41], the stage/memory occupancy of some major tables, such as the VM-NC mapping table, has already exceeded the resources of a single pipeline. Besides, other tables, such as SNAT/DNAT table for Internet access and meter tables for rate-limiting, will also have large memory footprints. Although the table sizes will shrink at the edge, a single Tofino pipeline still cannot hold all the converged core functions of the VPC network.



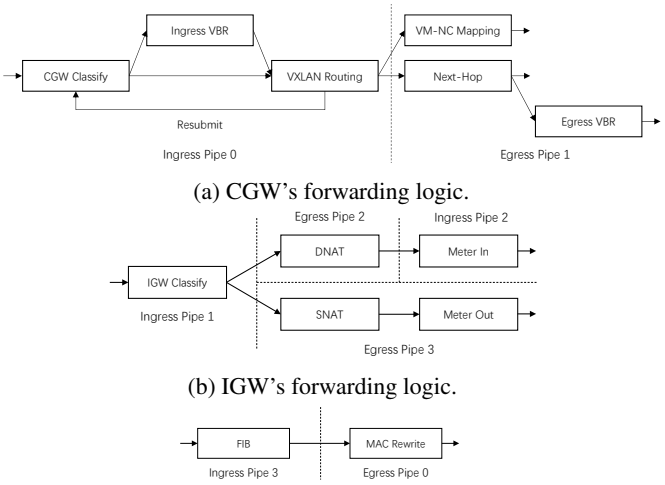
(a) LuoShen's pipeline layout. (b) Another layout option.

Figure 4: Two optional ways of pipeline folding.

To this end, we conduct pipeline folding to assemble the four independent pipelines into a 48-stage deep pipeline for CGW/IGW/SW convergence (as shown in Fig. 4a). The traffic of different cloud services will sequentially traverse through the deep pipeline and be processed at the corresponding stages. For example, the VM-VM (same VPC) traffic will go through Ingress Pipe 0, Egress Pipe 1, Ingress Pipe 1, Egress Pipe 3, Ingress Pipe 3, and Egress Pipe 0, while the Internet-VM traffic will go through Ingress Pipe 0, Egress Pipe 1, Ingress Pipe 1, Egress Pipe 2, Ingress Pipe 2, Egress Pipe 3, Ingress Pipe 3, and Egress Pipe 0. Notice that there is one explicit pipe branch at Ingress Pipe 1 with either Egress Pipe 2 or Egress Pipe 3 as the next-hop pipe to separately process Internet-VM or VM-Internet traffic. There is also an inexplicit pipe brach at Ingress Pipe 0 (not shown in Fig. 4a) with either Egress Pipe 1 or Ingress Pipe 0 itself as the next-hop pipe as some cloud services need to query the VXLAN routing table twice (*e.g.*, VM-to-VM traffic across VPCs). Last but not least, as the FIB in SW is embedded in Ingress Pipe 3, it will select the next-hop ports from Egress Pipe 0 according to the FIB lookup results (the 16 ports of Egress Pipe 0 are connected to the external destinations outside the Tofino such as XGW/SLB in CPU, SLB+ in FPGA, physical servers, Internet/remote regions through BSW, and on-premises IDCs).

Major tables in the pipeline. We show how the major tables of CGW, IGW and SW are distributed in the pipeline.

①CGW (Fig. 5a): CGW contains all the major tables of VGW, TGW and CSW. As the VXLAN routing table and the VM-NC mapping table are the two largest, we separate them into different pipes. To determine whether an incoming packet will be processed locally or bypass the current pipe, we place a CGW Classify at the front of Ingress Pipe 0. In this way, Internet traffic will directly bypass CGW (we detail the bypass logic later). The packet flows of different cloud services related to CGW are listed as follows. VM-VM (same VPC): CGW Classify->VXLAN Routing->VM-NC Mapping; VM-VM (across VPCs, same region): CGW Classify->VXLAN Routing(resubmit)->VM-NC Mapping; VM-Cross-region-VM (sender): CGW Classify->VXLAN Routing(resubmit)->Next-Hop; VM-Cross-region-VM (receiver): CGW Classify->VXLAN Routing(resubmit)->VM-NC Mapping; VM-IDC: CGW Classify->VXLAN Routing(resubmit)->Next-Hop->Egress VBR(VXLAN-to-VLAN); IDC-VM: CGW Classify->Ingress VBR(VLAN-to-VXLAN)->VXLAN Routing(resubmit)->VM-NC Mapping. Notice that we use Ingress/Egress VBR for VLAN-to-VXLAN translation and vice versa to deal with IDC traffic. Besides, we use Next-Hop to obtain the physical address of the remote device in either VM-Cross-region-VM (sender) or VM-IDC case. Finally, the resubmit ability of Tofino enables circular lookups of the VXLAN routing table in several cases.



(a) CGW's forwarding logic.

(b) IGW's forwarding logic.

(c) SW's forwarding logic.

Figure 5: Distribution of major tables in the pipeline.

②IGW (Fig. 5b): IGW contains SNAT for VM-Internet traffic and DNAT for Internet-VM traffic, as well as their rate-limiting tables (*i.e.*, Meter Out and Meter In). The packet flows are: VM-Internet: IGW Classify->SNAT->Meter Out; Internet-VM: IGW Classify->DNAT->Meter In. Similar to CGW Classify, IGW Classify is used to filter the traffic that should be processed locally. Besides, it will also decide whether the traffic should be sent to SNAT or DNAT according to packet header fields. Notice that Meter In occupies the entire pipe while Meter Out shares its pipe with the SNAT table. The reason is that, most rate-limiters for the VM-Internet direction are installed at vSwitches while all rate-limiters for the opposite direction are installed at the gateway, so that Meter Out is much smaller than Meter In at the gateway.

③SW (Fig. 5c): SW is responsible for underlay traffic forwarding according to the outer DIP. The possible destinations can be XGW/SLB in CPU, SLB+ in FPGA, physical servers, Internet/remote regions through BSW, and IDCs. SW conducts FIB lookup at Ingress Pipe 3 to select the next hop and then rewrites the MAC headers at Egress Pipe 0.

Pipe/table bypass logic. Although each packet will sequentially pass through CGW, IGW and SW, not all tables have to be queried, *e.g.*, Internet traffic will go through the CGW pipes but there is no need to query the VXLAN routing table. Hence, we can make an early judgment to determine whether the packet will be processed by the local pipe or even the local table to reduce unnecessary processing overhead. Usually, the judgment is placed at the front of a pipe, *e.g.*, in CGW Classify, we set flags in the metadata according to the VNI and DIP in the packet header to classify traffic into different cloud services and the flags can be carried by the metadata across pipeline stages. Fig. 6 shows the P4 code framework

```
struct metadata_t {
    bit<1> flag; /* whether to bypass the current pipe */
    bit<4> subflag; /* whether to bypass the current table */
    ...
}
control Ingress( ... ) {
    action tbl_ac1() { flag = 0; }
    action tbl_ac2() { flag = 1; subflag = 0; }
    action tbl_ac3() { flag = 1; subflag = 1; }
    ...
    table tbl {
        /* to distinguish different cloud services */
        key = { ... }
        /* to take different bypass actions */
        actions = {
            tbl_ac1; /* bypass the current pipe */
            tbl_ac2; /* enter the current pipe, query tb2 */
            tbl_ac3; /* enter the current pipe, query tb3 */
            ... } }
    table tb2 { ... }
    table tb3 { ... }
    ...
    apply {
        tbl.apply();
        if (flag == 1) {
            if (subflag == 0) { tb2.apply(); }
            else if (subflag == 1) { tb3.apply(); }
            ...
        }
        ... /* flag is 0, bypass the current pipe */
    } }
```

Figure 6: P4 code framework for pipe/table bypass.

for pipe/table bypass. Specifically, we use flag for pipe bypass and subflag for table bypass in the local pipe. For components that span across multiple pipes (*e.g.*, IGW), we can place separate judgment logic at the front of each pipe, or let the judgment results from the first pipe pass down to the subsequent pipes. The former consumes additional stage occupancy while the latter increases metadata bridge usage [41].

Rationale behind pipeline design.

①*Two optional ways of pipeline folding*. In designing the pipeline layout, we have also considered another option in Fig. 4b, which has separate pipelines folded for CGW and IGW, achieving 3.2Tbps total bandwidth. However, it has limitations: (1) Traffic balance is hard to achieve between two pipelines. (2) Table occupancy balance is also hard to guarantee between two pipelines. (3) Although more ports are exposed to servers, there is also tight coupling between the cloud services and the ports, which adds constraints for server placement. (4) Our selected pipeline layout maximally reuses the existing code by porting the entire IGW codebase from the existing P4-based IGW gateway [41] for rapid deployment.

②*Component sequence in the pipeline*. Someone may wonder why the sequence is CGW->IGW->SW. In fact, we can place either CGW or IGW in the front. However, the overlay devices (CGW and IGW) must be placed ahead of the underlay device (SW) because the gateway always conducts outer IP modification based on the VNI and the inner IP first (*e.g.*, via VXLAN route lookup), then uses the outer DIP to query the FIB in SW to determine the next-hop forwarding port.

③*Next-hop selection at the ingress pipe*. You may notice that all next-hop selection happens at the ingress pipe (*e.g.*, VXLAN routing table resubmit at Ingress Pipe 0, SNAT/DNAT selection at Ingress Pipe 1, FIB lookup in SW at Ingress Pipe 3). This is because we must rely on the TM of Tofino for packet switching and it is necessary to determine where the packet will go before it is sent to the TM.

④*Performance issue and the coping strategy*. The current pipeline layout also has limitations: (1) The original 4 parallel pipelines are folded into one, decreasing the bandwidth from 6.4Tbps to 1.6Tbps. As 4 ports are connected with the CPU and FPGA, the actual gateway bandwidth is only 1.2Tbps. (2) Lots of Tofino ports are used for internal loopback, thereby greatly reducing the number of ports for connecting physical servers. However, for edge clouds, as the 1.2Tbps throughput is more than enough and the server payload in a Cloud Box is restricted, the above-mentioned will no longer be a problem. In real deployment, hot standby is leveraged which doubles the available gateway bandwidth as well as server payload.

4.2 On-Chip Resource Budgeting

As Tofino has scarce SRAM/TCAM, pipeline stages, and PHV resources, we need to carefully budget the on-chip resource occupancy to fit multiple gateway functions into the chip.

Table placement for balanced resource occupancy. In LuoShen, forwarding tables are distributed in the 8 cascaded pipes based on the service logic as well as balanced pipeline resource occupancy. As mentioned in 4.1, the SRAM/TCAM memories in each pipeline stage are shared by the ingress pipe and the egress pipe. That is, for each pipeline, we can have different resource occupancy combinations like ingress-heavy, egress-heavy or balanced, but we can never have ingress-heavy and egress-heavy simultaneously. When fitting the tables into the Tofino, we also follow this principle to balance the resource occupancy of the 4 pipelines. For example, Pipeline 0 is ingress-heavy as its ingress pipe stores the VXLAN routing table, therefore, its egress pipe can only store very small tables for MAC rewrite. Similarly, Pipeline 1 is egress-heavy due to the VM-NC mapping table at Egress Pipe 1, Pipeline 2 is balanced because its ingress pipe has Meter In while its egress pipe has DNAT, Pipeline 3 is egress-heavy due to the SNAT and Meter Out at its egress pipe.

Stage occupancy compression. After pipeline folding, all tables of the cascaded CGW/IGW/SW are fit in the 48 stages. The stages are consumed mainly in two ways: (1) table dependency, (2) large table occupancy. The table dependency means the packet processing behavior of the current stage depends on the table lookup results of the previous stage, *e.g.*, CGW Classify occupies the first stage of Pipeline 0 to set bypass flags for subsequent stages. Clearly, in a hyper-converged gateway, table dependencies occur more often. The large table occupancy means the large table that cannot be fit in one stage will span across multiple stages. In LuoShen, for (1), we rely on the P4 compiler to generate the dependency graph; for (2), we shrink some large tables (*e.g.*, the VM-NC mapping table) to reduce their stage occupancy for edge clouds.

PHV usage optimizations. PHV is a set of containers that carry the headers and metadata along the pipeline. The first PHV optimization is to reduce the metadata bridging as much as possible by meticulous table placement because metadata bridging not only decreases the switching throughput through the TM [41] but also consumes more PHV resources at both

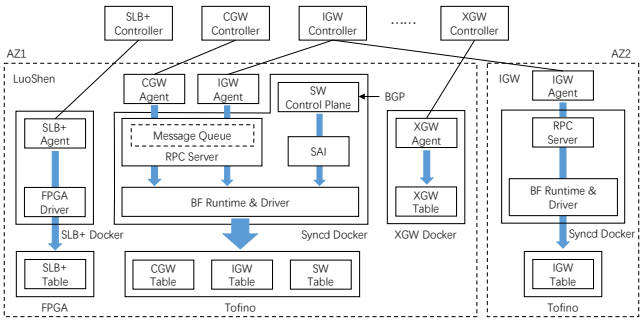


Figure 7: Multi-component table configuration channels.

ingress pipe and egress pipe. The second PHV optimization is to overlay two or more header/metadata fields in the same PHV container if their lifetimes do not intersect by using the *pa_alias* pragma to instruct the compiler. Moreover, we break the large header/metadata field into smaller pieces of the same size intentionally to increase the possibility of PHV sharing.

5 Control Plane

Resource isolation. Multiple components run on LuoShen's CPU. They are (1) DPDK-based forwarding instances such as XGW and SLB, (2) control plane agents of the data plane components in Tofino (CGW/IGW/SW), FPGA (SLB+) and CPU (XGW/SLB). Although the underlying CPU resources are shared, these components have different performance requirements (*e.g.*, line-rate forwarding) and do not want to be interrupted by others. To achieve performance isolation, we dockerize [38] these components and bind XGW and SLB to dedicated CPU cores. As the number of CPU cores is limited, we also allow components (*e.g.*, CGW and IGW agents) to share the CPU cores while using cgroups [46] for resource isolation. Except for CPU, resource isolation for memory/disk is also needed to contain memory leaks and core dump files.

Multi-component table configuration. Unlike the single-role gateways, LuoShen converges multiple components in its data plane, thereby requiring multiple table configuration channels (Fig. 7). For each channel, a control plane agent at the CPU receives table update requests pushed from its remote controller and installs them into the corresponding data plane component through the underlying interface (*e.g.*, BF Runtime). For rapid deployment of LuoShen, we reuse the agent code directly from the single-role gateway so that the remote controller can talk directly to LuoShen without any modification (*e.g.*, in Fig. 7, the IGW controller controls LuoShen and IGW). The channels to Tofino, FPGA and CPU are physically isolated to avoid table configuration collisions. For the Tofino, the channel to CGW/IGW and the channel to SW are also separate (*i.e.*, through RPC and SAI). Although CGW and IGW share the same channel, the complete separation of their tables in the Tofino pipeline makes their table configuration entirely lockless. Besides, to speed up table configuration, we use batch to assemble multiple table update requests and issue them in one shot to the BF Runtime.

Inter-component BGP peering. LuoShen converges the role-splitting VPC network infrastructure of the public cloud into one device, where internal components still need to communicate with each other (*e.g.*, SW needs to route traffic to the next-hop components). To exchange the reachability information between components, we set up BGP speakers at the control plane for inter-component BGP peering so that a component can learn the routes to others and it can also advertise its reachability to others. Specifically, we set up separate BGP speakers for XGW, SLB, SLB+ with both learning and advertising capabilities. For CGW, IGW, SW, we set up one BGP speaker for all of them. For CGW and IGW, as there is only one path in the pipeline, there is no need for them to learn BGP routes and their agents only perform advertising. For SW, as it is the final component in the pipeline, its control plane only needs to learn the routes for next-hop selection. With BGP peering, LuoShen achieves high availability based on component-level ECMP load balancing and fast failure recovery, which are discussed in §A due to page limitation.

6 LuoShen's Performance

We illustrate LuoShen's performance under pressure test and in production. The pressure test topology is shown in §B.

On-chip memory occupancy. Fig. 8 shows the memory usage in ingress/egress pipes. Generally, LuoShen consumes more SRAM than TCAM because most major tables (*e.g.*, VM-NC, SNAT/DNAT, meters, counters) are based on exact match. Besides, as the Tofino contains more SRAM than TCAM, we use ALPM [51] to reduce the TCAM usage at the cost of additional SRAM usage [41]. Except for the VXLAN routing table, Classify tables, ACLs, and FIB in SW will also consume TCAM. Fig. 9 and Fig. 10 show how we achieve balanced pipeline resource occupancy via meticulous table placement. For Pipeline 0, we place the VXLAN routing table in the later stages in the ingress pipe as it has dependencies with the tables in its front (*e.g.*, CGW Classify). As each stage is shared by the ingress and egress, we have to place the tables for MAC rewrite in the early stages in the egress pipe for balanced memory usage. For Pipeline 2, the pipeline resources are shared in a good balance by the ingress (Meter In) and egress (DNAT). Fig. 11 shows the memory usage of CGW/IGW/SW. The SRAM/TCAM occupancy is exactly as expected. However, even in a hyper-converged gateway, the memories are not fully exhausted. The reason is that table dependencies make stages exhausted earlier than memories. Fig. 12 shows the maximum PHV usage in different pipelines. After PHV optimizations, the PHV has not been exhausted in all pipelines. Pipeline 2 has the lowest PHV usage since it contains only two major tables (*i.e.*, DNAT and Meter In).

Performance under pressure test. Fig. 13 shows table configuration speedup via batching. As CGW and IGW share the same download channel, the improved table configuration efficiency will not only reduce the gateway's cold start time but also alleviate download channel congestion. Fig. 14 shows the throughput and latency of the VM-to-VM traffic within the same VPC. LuoShen can achieve 1.2Tbps line-rate forwarding with 256B packets and the latency is bounded within

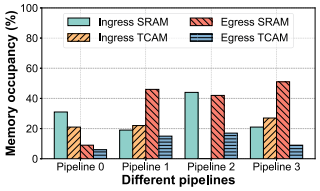


Figure 8: SRAM/TCAM usage in ingress/egress pipes.

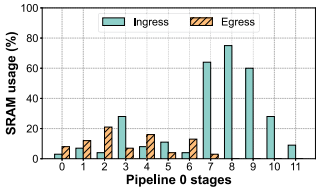


Figure 9: SRAM usage distribution over stages (Pipeline 0).

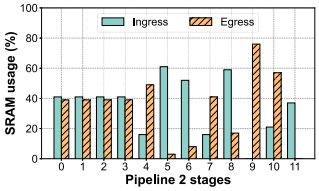


Figure 10: SRAM usage distribution over stages (Pipeline 1).

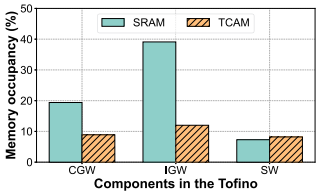


Figure 11: SRAM/TCAM usage of CGW/IGW/SW.

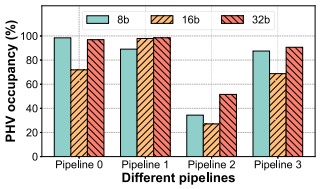


Figure 12: PHV usage in different pipelines (the worst case).

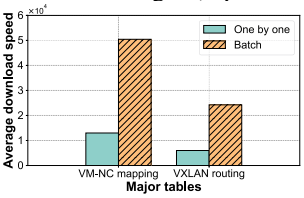


Figure 13: Table configuration speedup via batching.

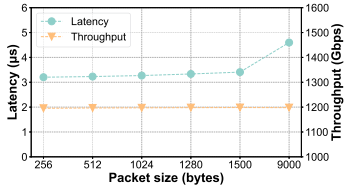


Figure 14: Throughput/latency of VM-to-VM traffic (same VPC).

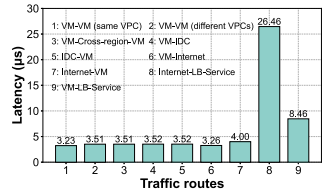


Figure 15: Latency of different traffic routes (512B packets).

5μs even with 9000B packets. Fig. 15 shows the latency of different cloud traffic routes with 512B packet size. The traffic of VM-VM (same VPC) and VM-Internet has the lowest latency due to the shortest route inside the Tofino (only through 6 pipes) without a resubmit at Ingress Pipe 0. The traffic of cross-VPC, cross-region and IDC has a slightly higher latency due to the resubmit operation. The traffic of Internet-VM has an even higher latency due to its longer route through 8 pipes. The traffic of the remaining cloud services will be routed to the external CPU and FPGA, producing the longest latency. Fig. 16 shows the throughput of different cloud traffic routes with 512B packet size. The traffic traversing through the external CPU and FPGA has the lowest throughput, exactly as expected. However, we notice that the traffic of VM-IDC and VM-Internet also has slightly reduced throughput. The reason is that they both experience decapsulation of outer headers.

Performance in production. Fig. 17 shows the converged traffic throughput at an edge site, which is around 50Gbps. As edge clouds have just been rolled out, most edge sites do not have heavy traffic. However, we still select the P4-centric architecture as it not only addresses the extreme cases but also reserves a large performance margin for future traffic growth, which makes the architecture durable. Fig. 18 shows the traffic throughput of different cloud services of the edge site. It can be inferred that most edge cloud traffic is IDC/Internet-related. Most of the VM-to-VM traffic within the same VPC is shortcut by the underlay network due to route caching. Cross-VPC traffic is currently very limited. Fig. 19 shows the CPU utilization of different components. SLB has the highest CPU usage as a data plane function. The control plane agents of CGW/IGW/SW also consume considerable CPU time.

Advantages of LuoShen at the edge. We compare LuoShen with the role-splitting gateway architecture in the public cloud in upfront cost, deployment size and power consumption. We assume that the gateway/load balancer/switch clusters in the public cloud will be reduced to separate devices in edge clouds with one device for each role. Table 2 shows that LuoShen reduces the upfront cost, deployment footprints and power consumption by 75%, 87% and 60%, respectively. Due to page limitation, we discuss the calculation in §C.

Table 2: LuoShen vs role-spliting in cost, size and power.

	Cost (unit)	Size (U)	Power (W)
LuoShen	15	2	∼1000
Role-splitting	61	15	>2500

7 Experiences and Lessons

Step-by-step deployment. To decompose the complexity from infrastructure convergence, we deploy LuoShen in a step-by-step way in production. We deploy the components in the Tofino first since CGW and IGW cover the majority of cloud network functions while SW provides the next-hop selection ability. XGW/SLB/SLB+ are implemented by reusing the existing x86 devices in the public cloud. As CGW/IGW/SW are highly coupled, we have to deploy them together in one pass. After they become more stable, we start to add XGW, SLB and SLB+ one by one for complete deployment.

Function upgrade. As Tofino, CPU and FPGA are physically separated, they can be upgraded independently. By contrast, as CGW, IGW and SW are highly coupled, we have to upgrade them as a whole, which will involve cross-team collaboration as they are managed by different teams. The function isolation issue in Tofino has also been discussed in [19, 53]. In 2018, Barefoot announced Tofino Fast Refresh [14] to address the hot upgrade issue. The feature can reset an entire P4 pipeline within 50ms. However, as LuoShen deeply relies on pipeline folding with one component spanning across pipelines and two components possibly sharing the same pipeline, simply using Tofino Fast Refresh for component upgrade may have consistency issues. For example, when CGW resets Pipeline 0 and Pipeline 1 for an upgrade, the state of SW and IGW in the same two pipelines will disappear for a short time and their table query/update requests on the fly will be discarded.

Capacity expansion. The gateway capacity expansion includes the upgrade of performance and table size. For performance, CPU is more likely to become the system bottleneck. Some of the solutions we've tried include (1) adjusting the number of CPU cores used by different services, (2) migrating workloads from LuoShen to external x86 servers, (3) horizon-

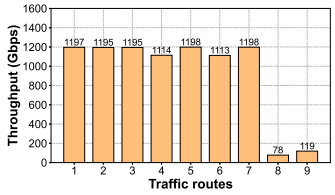


Figure 16: Throughput of different traffic routes (512B packets).

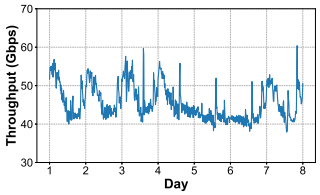


Figure 17: Converged traffic throughput at an edge site.

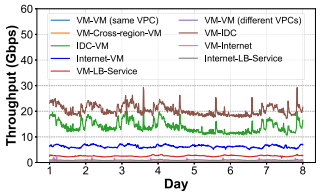


Figure 18: Traffic throughput of different cloud services.

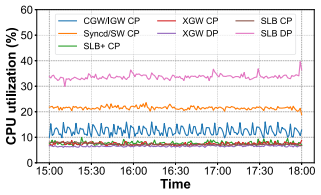


Figure 19: CPU utilization of different components.

tal scaling with more LuoShen gateways. In real deployment, we prefer (3) for stability and ease of maintenance. For table size, we borrow the horizontal table splitting technique from Sailfish [41] to share large tables among more gateways.

Failure isolation. In the public cloud, different gateway clusters are physically separated. In each cluster, tenants' entries are horizontally split into multiple gateways. Therefore, failures will be isolated within a gateway. However, as each gateway is associated with a huge number of tenants, the blast radius of a failure will still be large. In edge clouds, LuoShen serves all tenants miles away. That is, the failure of LuoShen will affect all tenants at the edge site, although the number will not be so large. In the Tofino, as CGW/IGW/SW are cascaded in a folded pipeline, the stability of each component is critical and we reuse the mature code from the single-role gateways. At the CPU, we rely on docker and cgroup for failure isolation. Finally, hot standby is our last resort to guarantee a failsafe.

Telemetry and debugging. As packet flows of different cloud services converge at LuoShen, for network-wide telemetry, we should probe all traffic routes in the gateway [59]. To achieve this, we generate probes with pre-defined header fields for all cloud service coverage. For network anomaly debugging, we need to pinpoint the exact anomaly locations. To achieve this, we collect and export telemetry data to the CPU at each component/pipe/stage along the anomaly forwarding path.

Elastic NFV deployment. For LuoShen, if we deploy NFV instances in its CPU, the elasticity will be restricted as the CPU has already been excessively used (with XGW, SLB and control plane agents). But, if we deploy NFV instances in x86 servers, the CPU cores for tenants' VMs will be occupied (some tenants are concerned about this as the server payload is valuable at the edge). We develop an NFV framework that can autoscale across LuoShen's boundary by horizontally expanding table entries onto x86 servers during peak workloads.

8 Related Work

Building hybrid CPU/ASIC/FPGA network systems is a common tactic [9, 11, 35, 41, 57] to combat traffic growth that goes far beyond Moore's law [45]. However, few systems succeed in fitting the entire multi-tenant cloud network infrastructure with multi-service packet flows into a 2U box deployed in production. For example, ServerSwitch [35] takes advantage of both commodity servers and switching ASICs to perform flexible and performant underlay traffic processing in DCNs. Tiara [57] proposes a hardware-accelerated L4 load balancer with a 3-tier architecture of P4/FPGA/CPU. While Tiara fits the SLB function for the public cloud into a 5U box, LuoShen fits the entire VPC network infrastructure (including SLB) into a 2U box tailored for edge clouds. Sailfish [41] uses Tofino to accelerate the single-role cloud gateways (*e.g.*, VGW). While Sailfish and LuoShen both use pipeline folding to provide sufficient stage resources, LuoShen considers additional constraints. In addition to fitting table entries within Tofino, it must also consolidate multi-service packet flows, achieve equivalent implementation of underlay/overlay devices, expose ports to interact with external CPU/FPGA, and maximize codebase reuse. Bluebird [11] uses Tofino to accelerate the bare-metal cloud services which have similar packet flows with our VM-IDC, IDC-VM scenarios with VXLAN-to-VLAN translation and vice versa. Comparatively, LuoShen is an "all-in-one" gateway with a more sophisticated pipeline layout to consolidate almost all cloud network functions.

Different from our centralized gateway model, other cloud vendors distribute the VPC network functions to end-hosts [12, 17, 20–22, 31, 33]. For example, VFP [21] installs virtual routing, tunnel encap/decap, and load balancing into a host stack, which is scalable and fault-tolerant as there is no single point of failure. The host stack performance can further be accelerated by SmartNICs [22] or by a remote DPU pool [12] to deal with high-bandwidth traffic bursts. Such distributed cloud network architecture could also be a good fit for edge clouds. Considering the consistency with our existing infrastructure, LuoShen takes an alternative approach with different design tradeoffs. It offloads almost all VPC network functions to the gateway to maximally free up valuable CPU cores for tenants' VMs. SmartNIC acceleration is not necessary which saves upfront cost. The IDC/cross-region traffic is also handled by the same gateway which saves cost and space. To address single point of failure, LuoShen relies on Tofino to absorb traffic bursts and reuses mature code for stability.

Some vendors build white box server switches without service logic (*e.g.*, Accton's CSP-7551 [4]). LuoShen's idea can be extended to them to build other hyper-converged systems.

9 Conclusion

We propose LuoShen, a hyper-converged gateway for edge clouds, which fits the entire VPC network infrastructure of the public cloud into a 2U server switch with a P4-centric architecture. To adapt to the new design constraints of performance, costs and deployment footprints, we rearchitect the data plane and control plane. LuoShen has been deployed in Alibaba Cloud for over two years at hundreds of edge sites.

Acknowledgements: We thank the anonymous reviewers and our shepherd Anirudh Sivaraman for constructive feedback.

References

[1] Alibaba cloud launches cloud box - extending public cloud services to local devices. https://www.alibabacloud.com/blog/alibaba-cloud-launches-cloud-box---extending-public-cloud-services-to-local-devices_596688, 2020.

[2] Alibaba cloud launches local regions. https://infrastructure.aliyun.com/local-regions, 2021.

[3] Alibaba cloudbox - product page. https://www.aliyun.com/product/ecs/cloudbox, 2021.

[4] Csp-7551, hyper network appliance. https://www.accton.com/product-csp-7551/, 2021.

[5] Tofino: P4-programmable ethernet switch asic that delivers better performance at lower power. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html, 2021.

[6] Alibaba cloud's global infrastructure. https://www.alibabacloud.com/global-locations, 2022.

[7] Cascade lake: Overview. https://www.intel.com/content/www/us/en/products/platforms/details/cascade-lake.html, 2022.

[8] The journey towards predictable network in alibaba cloud. https://opennetworking.org/wp-content/uploads/2022/05/Dennis-Cai-Final-Slide-Deck.pdf, 2022.

[9] Mina Tahmasbi Arashloo, Pavel Shirshov, Rohan Gandhi, Guohan Lu, Lihua Yuan, and Jennifer Rexford. A scalable vpn gateway for multi-tenant cloud services. *ACM SIGCOMM Computer Communication Review*, 48(1):49–55, 2018.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[11] Manikandan Arumugam, Deepak Bansal, Navdeep Bhatia, James Boerner, Simon Capper, Changhoon Kim, Sarah McClure, Neeraj Motwani, Ranga Narasimhan, Urvish Panchal, et al. Bluebird: High-performance {SDN} for bare-metal cloud services. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 355–370, 2022.

[12] Deepak Bansal, Gerald DeGrace, Rishabh Tewari, Michal Zygmunt, James Grantham, Silvano Gai, Mario Baldi, Krishna Doddapaneni, Arun Selvarajan, Arunkumar Arumugam, et al. Disaggregating stateful network functions. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1469–1487, 2023.

[13] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 5–16. IEEE, 2015.

[14] Antonin Bas. Leveraging stratum and tofino fast refresh for software upgrades. *Accessed: Jul*, 4:2021, 2018.

[15] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[16] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[17] 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.

[18] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 523–535, 2016.

[19] Yong Feng, Zhikang Chen, Haoyu Song, Wenquan Xu, Jiahao Li, Zijian Zhang, Tong Yun, Ying Wan, and Bin Liu. Enabling in-situ programmability in network data plane: From architecture to language. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 635–649, 2022.

[20] DANIEL FIRESTONE. A virtual switch platform for host sdn in the public cloud. *USENIX Open Access Policy*, page 6.

[21] Daniel Firestone. {VFP}: A virtual switch platform for host {SDN} in the public cloud. In *14th USENIX*

Symposium on Networked Systems Design and Implementation (NSDI 17), pages 315–328, 2017.

[22] 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.

[23] Xenofon Foukas and Bozidar Radunovic. Concordia: Teaching the 5g vran to share compute. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 580–596, 2021.

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

[25] Stephen D Goglin and Linden Cornett. Flexible and extensible receive side scaling, September 1 2009. US Patent 7,584,286.

[26] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[27] Jesse Gross, Ilango Ganga, and T. Sridhar. Geneve: Generic Network Virtualization Encapsulation. RFC 8926, November 2020.

[28] Rolf Harms and Michael Yamartino. The economics of the cloud. *Microsoft whitepaper, Microsoft Corporation*, 3:157, 2010.

[29] Antonios Katsarakis, Zhaowei Tan, Matthew Balkwill, Bozidar Radunovic, Andrew Bainbridge, Aleksandar Dragojevic, Boris Grot, and Yongguang Zhang. rvnf: Reliable, scalable and performant cellular vnfs in the cloud. Technical report, Technical Report MSR-TR-2021-7, Microsoft, 2021.

[30] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. Tea: Enabling state-intensive network functions on programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 90–106, 2020.

[31] 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.

[32] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14, 2010.

[33] Anthony Liguori. The nitro project–next generation aws infrastructure. In *Hot Chips: A Symposium on High Performance Chips*, 2018.

[34] Yunzhuo Liu, Hao Nie, Hui Cai, Bo Jiang, Pengyu Zhang, Yirui Liu, Yidong Yao, Xionglie Wei, Biao Lyu, Chenren Xu, et al. X-plane: A high-throughput large-capacity 5g upf. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2023.

[35] Guohan Lu, Chuanxiong Guo, Yulong Li, Zhiqiang Zhou, Tong Yuan, Haitao Wu, Yongqiang Xiong, Rui Gao, and Yongguang Zhang. {ServerSwitch}: A programmable and high performance platform for data center networks. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[36] Jianyuan Lu, Tian Pan, Shan He, Mao Miao, Guangzhe Zhou, Yining Qi, Shize Zhang, Enge Song, Xiaoqing Sun, Huaiyi Zhao, et al. Cloudsentry: Two-stage heavy hitter detection for cloud-scale gateway overload protection. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[37] Mallik Mahalingam, Dinesh 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. Technical report, 2014.

[38] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[39] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[40] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. Netlord: a scalable multi-tenant network architecture for virtualized datacenters. *ACM SIGCOMM Computer Communication Review*, 41(4):62–73, 2011.

[41] 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.

[42] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. Ananta: Cloud scale load balancing. *ACM SIGCOMM Computer Communication Review*, 43(4):207–218, 2013.

[43] Zachary NJ Peterson, Mark Gondree, and Robert Beverly. A position paper on data sovereignty: The importance of geolocating data in the cloud. In *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 11)*, 2011.

[44] 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.

[45] Lawrence G Roberts. Beyond moore's law: Internet growth trends. *Computer*, 33(1):117–119, 2000.

[46] Rami Rosen. Resource management: Linux kernel namespaces and cgroups. *Haifux, May*, 186:70, 2013.

[47] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[48] Enge Song, Nianbing Yu, Tian Pan, Qiang Fu, Liang Xu, Xionglie Wei, Yisong Qiao, Jianyuan Lu, Yijian Dong, Mingxu Xie, et al. Mimic: Smartnic-aided flow backpressure for cpu overloading protection in multi-tenant clouds. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2022.

[49] George J Stigler. The economies of scale. *The Journal of Law and Economics*, 1:54–71, 1958.

[50] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[51] Henry Wang. Algorithmic longest prefix matching in programmable switch, December 17 2019. US Patent 10,511,532.

[52] Jianyu Wang, Jianli Pan, Flavio Esposito, Prasad Calyam, Zhicheng Yang, and Prasant Mohapatra. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–23, 2019.

[53] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. Isolation mechanisms for {High-Speed}{Packet-Processing} pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1289–1305, 2022.

[54] Chengkun Wei, Xing Li, Ye Yang, Xiaochong Jiang, Tianyu Xu, Bowen Yang, Taotao Wu, Chao Xu, Yilong Lv, Haifeng Gao, et al. Achelous: Enabling programmability, elasticity, and reliability in hyperscale cloud networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 769–782, 2023.

[55] Timothy Wood, Prashant J Shenoy, Alexandre Gerber, Jacobus E van der Merwe, and Kadangode K Ramakrishnan. The case for enterprise-ready virtual private clouds. In *HotCloud*, 2009.

[56] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*, pages 331–344. IEEE, 2019.

[57] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1345–1358, 2022.

[58] Xiantao Zhang, Xiao Zheng, Zhi Wang, Hang Yang, Yibin Shen, and Xin Long. High-density multi-tenant bare-metal cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 483–495, 2020.

[59] Shunmin Zhu, Jianyuan Lu, Biao Lyu, Tian Pan, Chenhao Jia, Xin Cheng, Daxiang Kang, Yilong Lv, Fukun Yang, Xiaobo Xue, et al. Zoonet: a proactive telemetry system for large-scale cloud networks. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, pages 321–336, 2022.

Appendices

A High Availability with Hot Standby

In a production environment, we deploy at least two LuoShen gateways in a hot-standby mode to achieve high availability, as shown in Fig. A1. The same components of the two LuoShen gateways will advertise the same VIP to the outside so that the upstream switches/routers can conduct ECMP load balancing to split the incoming traffic equally between the two components. With such design, we can easily scale out our gateway system to handle the edge cloud traffic growth in the future. For example, in Fig. A1, two CGW components advertise the same VIP to the upstream BSW, which will conduct ECMP load balancing on the traffic towards CGW. Except for traffic load balancing, the same components of the two LuoShen gateways can also back up for each other for fast failure recovery. For example, in Fig. A1, two IGW components advertise the same VIP to the upstream BSW, which will conduct ECMP load balancing on the traffic towards IGW. However, at a certain time, if there is a link or component failure of the IGW in one gateway, it will trigger BGP route withdrawal in the upstream BSW. After that, traffic will be forwarded by the upstream BSW to the remaining IGW. According to our measurement, the BGP route withdrawal will be quickly completed in milliseconds during a link failure.

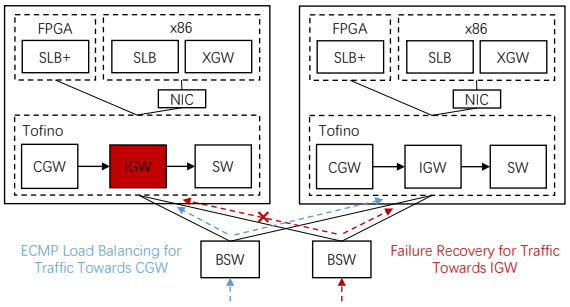


Figure A1: In production, LuoShen achieves load balancing and high availability with hot-standby deployment.

B Performance Test Topology

In order to pressure test LuoShen's performance, we build a test topology as shown in Fig. A2. As mentioned in §4.1, the Tofino chip in LuoShen only exposes the 16 ports of its Pipeline 0 after pipeline folding and 4 of them are connected with the CPU and FPGA. Therefore, we use a traffic generator to inject 1.2Tbps traffic through 12 optical fibers into the remaining 12×100G ports. By changing the traffic header fields (*e.g.*, VNI, DIP), we can conduct pressure tests on different traffic routes inside LuoShen for different cloud services.

C Calculation of Cost, Size and Power

When deploying the role-splitting gateway architecture of the public cloud at the edge, due to the reduced traffic volumes at the edge, we reduce the original role-based clusters of devices

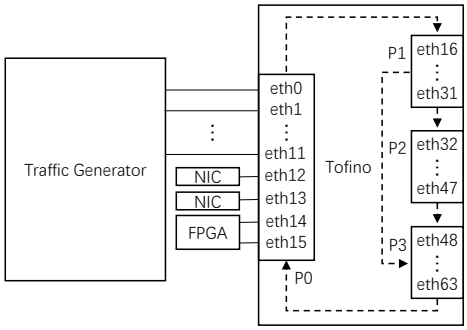


Figure A2: Performance test topology.

to multiple devices accordingly, with one device for a particular cloud function. In this way, we will have a device list including an x86 server attached with an FPGA (2U) to implement SLB+, two x86 servers (2*2U) to implement XGW and SLB, three P4-based gateways (3*2U) to implement VGW, IGW and TGW, three switches (3*1U) to implement CSW, LSW and the original SW. Then, we calculate the cost, size and power of LuoShen and the role-splitting architecture, respectively. ① Upfront cost: As the exact cost numbers of FPGA, x86 server, P4-based gateway and LuoShen are confidential, we normalize them to 1:10:10:15 according to our experience (for the sake of simplicity, we omit the cost of switches). Accordingly, the cost of LuoShen is 15 while the cost of the role-splitting is (1+10)+2*10+3*10=61. ② Deployment footprints: LuoShen will occupy 2U while the role-splitting will occupy 2U+2*2U+3*2U+3*1U=15U. ③ Power consumption: Generally, an FPGA consumes 100W [11, 57], an x86 server consumes 500W [57] and its CPU consumes 200W [7], a P4-based gateway consumes 300W [39]. Therefore, the power consumption of the role-splitting architecture is at least (100W+500W)+2*500W+3*300W=2500W (we still omit the power consumption of switches). LuoShen contains one P4 switch, two CPUs and one FPGA. Besides, the optical modules, fans, disks will additionally consume around 200W. Therefore, the power consumption of LuoShen is around 300W+2*200W+100W+200W=1000W.

D Acronym Definition

Table A1: Definition of vendor-specific acronyms.

Acronym	Expansion
SLB	Server Load Balancer
XGW	eXtendable Gateway
IGW	Internet Gateway
VGW	Virtual Private Gateway
TGW	Transit Gateway
CGW	Cloud Gateway
vSwitch	Virtual Switch
NC	Node Controller
SW	Switch
CSW	Customer Switch
LSW	Integrated Access Switch
BSW	Border Switch
IDC	Internet Data Center
VBR	Virtual Border Router