Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches

Tian Pan[†], Nianbing Yu[†], Chenhao Jia[†], Jianwen Pi[†], Liang Xu[†], Yisong Qiao[†], Zhiguo Li[†], Kun Liu[†], Jie Lu[†], Jianyuan Lu[†□], Enge Song[†], Jiao Zhang[*], Tao Huang[*], Shunmin Zhu[★†□]
[†]Alibaba Group [*]Purple Mountain Laboratories [★]Tsinghua University

ABSTRACT

The cloud gateway is essential in the public cloud as the central hub of cloud traffic. We show that horizontal scaling of software gateways, once sustainable for years, is no longer future-proof facing the massive scale and rapid growth of today's cloud. The root cause is the stagnant performance of the CPU core, which is prone to be overloaded by heavy hitters as traffic growth goes far beyond Moore's law. To address this, we propose *Sailfish*, a cloud-scale multi-tenant multi-service gateway accelerated by programmable switches. The new challenge is that large forwarding tables due to multi-tenancy cannot be fit into the limited on-chip memories. To this end, we devise a multi-pronged approach with (1) hardware/software co-design for table sharing, (2) horizontal table splitting among gateway clusters, (3) pipeline-aware table compression for a single node. Compared with the x86 gateway of a similar price, Sailfish reduces latency by 95% (2μs), improves throughput by more than 20x in bps (3.2Tbps) and 71x in pps (1.8Gpps) with packet length < 256B. Sailfish has been deployed in Alibaba Cloud for more than two years. It is the first P4-based cloud gateway in the industry, of which a single cluster carries dozens of Tbps traffic, withstanding peak-hour traffic in large online shopping festivals.

CCS CONCEPTS

• **Networks** → **Data center networks**; **Intermediate nodes**; **Programmable networks**.

KEYWORDS

virtual private cloud, cloud gateways, programmable data plane, forwarding table compression

ACM Reference Format:
Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, Enge Song, Jiao Zhang, Tao Huang, Shunmin Zhu. 2021. Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3452296.3472889

[□]Co-corresponding authors.

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472889

1 INTRODUCTION

The public clouds provide customers with the ability to use on-demand, scalable computing resources (*e.g.*, servers, storage, networking) to achieve agility, cost efficiency and business continuity based on cloud resource pooling and autoscaling [36]. Worldwide end-user spending on public cloud services has grown persistently over years [12] and the global pandemic has reinforced this trend [13]. To meet the increasing cloud usage demands, public cloud vendors, such as Amazon, Microsoft, Google and Alibaba, have invested lots of cutting-edge research and development efforts into addressing the performance, scalability, elasticity, stability and reliability of the globally distributed cloud infrastructures.

A cloud gateway is a packet forwarding device, which provides performant and reliable connectivity to globally distributed cloud resources in a multi-tenant environment [8, 25]. To satisfy the resource isolation requirements from customers, public cloud vendors build virtual private clouds (VPCs) [41] within the public cloud through overlay network protocols such as VXLAN [27]. Virtual machines (VMs) within the same VPC can communicate with each other while VMs belonging to different VPCs are isolated and transparent from each other. Sometimes, a VM in one VPC needs to communicate with another VM in a different VPC within the same region or in a remote region. In other cases, a VM in one VPC wants to access the public Internet or the resources inside the enterprise's local data centers (IDCs). The cloud gateway addresses these inter-VM and cross-region communication requirements and constructs tunnels if the communication needs to cross the VPC boundary.

As the *central hub* of the east-west traffic (VM-VM), the south-north traffic (VM-Internet) and the IDC/cross-region traffic (VM-IDC/Cross-region) of the entire cloud, the gateway plays a critical role in the public cloud infrastructure. The cloud has experienced a violent traffic growth over the years. Within data centers, servers' Ethernet transfer rates grow rapidly to 400GbE and beyond [35]. In 2019, Alibaba put the entire e-commerce business onto the public cloud [6], contributing to dozens of Tbps traffic for its cloud gateway. Therefore, the performance of the cloud gateway at the traffic aggregation point is expected to scale up gracefully to cope with the explosive traffic growth, preventing it from becoming the system choke point. Besides, the stability of the cloud gateway is also important for 24/7 uninterrupted cloud service delivery.

In addition to the forwarding performance, the cloud gateway needs to carry a large number of stateless and stateful forwarding tables for diverse cloud services. Specifically, due to the *multi-tenancy* in the public cloud, the forwarding table entry size is large containing both the VPC identifier and the destination address while the table entry number is huge considering the huge number of VPCs and VMs in the public cloud [8]. In Alibaba Cloud, a single cloud region can host millions of VPCs and millions of VMs.

Sometimes, a top customer can purchase millions of VMs even in a single VPC. In general, the cloud gateway differs from traditional network devices in terms of forwarding logic, forwarding throughput, forwarding table size and forwarding table variety.

To handle the growing workloads, horizontal scaling (*i.e.*, scale-out) of a single node to multiply the processing capability is common wisdom in the cloud [15, 16, 18, 31]. Over the years, x86-based software gateways were previously deployed in clusters in Alibaba Cloud to battle the rapid growth of cloud traffic. Benefiting from kernel-bypass techniques [9, 28], hardware upgrades and continuous system tweaks, these x86 boxes serve well and prosper for many years. However, the scale and growth rate of Alibaba Cloud today make such horizontal scaling unsustainable. The reason is three-fold. First, horizontal scaling increases the CapEx (*i.e.*, hardware acquisition cost) and OpEx (*i.e.*, operating expenses such as engineers consumed in system troubleshooting) of the entire cluster as well as the management and maintenance complexity since hundreds of x86 nodes are needed in a single cluster. Second, due to the weak performance of a single CPU core, the CPU core is likely to be overloaded due to traffic bursts brought by heavy-hitter flows [37], which further affects the performance isolation for multiple tenants and deteriorates the service-level agreements (SLAs). The CPU overload problem also compels system engineers to pre-allocate even more processing headroom for each CPU core, which in turn lowers the gateway cluster utilization and further increases the CapEx and OpEx. Last but not least is the slowdown of the performance improvement of a single CPU core in recent years [39]. It is well acknowledged that Moore's law goes far behind Internet traffic growth [34]. However, the performance improvement of a single CPU core is even slower than Moore's law. That is to say, the weak single CPU core will be persistently challenged by more and more heavy-hitter flows in the foreseeable future.

To address this performance issue, we propose *Sailfish*, an accelerated cloud-scale multi-tenant multi-service gateway with programmable switching ASICs. Designing Sailfish was challenging because the large forwarding tables due to multi-tenancy in the public cloud cannot be completely fit into the limited on-chip memories of the programmable switching ASICs (*e.g.*, Tofino) [7] and the memory space of those ASICs are not as flat as those of the x86 nodes [11]. To this end, we propose a multi-pronged approach. First, we conduct a hardware and software co-design for table sharing, putting a small number of more stable forwarding tables in hardware to cover the majority of traffic generated by frequently invoked cloud services, and leaving the remaining more volatile tables or very large stateful tables to software. Second, we propose horizontal table splitting among gateway clusters, which improves the scalability and fault isolation, reducing the maintenance complexity. Third, for a single node, we further conduct pipeline-aware table compression on the Tofino chip by the techniques such as pipeline folding, table splitting between pipelines, table mapping across pipelines, memory resource pooling, TCAM conservation and table entry compression. The single-node table compression increases the number of entries carried in one cluster, and thus reduces the number of clusters as well as the hardware acquisition/maintenance cost. More than two years of Sailfish deployment experiences about cluster construction, cluster management, disaster recovery as well as the lessons we've learned are also shared.

Our major contributions are summarized as follows.

- We propose the world's first multi-tenant multi-service cloud gateway based on programmable switches and disclose the technical details as well as deployment experiences in depth.
- We propose solutions that include conducting hardware and software co-design for table sharing, distributing table entries horizontally among gateway clusters and focusing on single-node capacity scale-up to resolve the memory shortage problem of the currently used programmable ASICs.
- On a single gateway, we carry out table compression that decreases SRAM occupancy by 38% and TCAM occupancy by 96% in the IPv4 scenario. In the IPv6 scenario, it decreases SRAM occupancy by 85% and TCAM occupancy by 98%.
- Compared with an x86 gateway node of roughly the same unit price, Sailfish lowers the latency by 95% (2μs), improves the throughput by more than 20x in bps (3.2Tbps) and 71x in pps (1.8Gpps) with packet length < 256B. Compared with the x86 gateway clusters, Sailfish reduces the total hardware acquisition cost by more than 90% for a region. Sailfish has been deployed in Alibaba Cloud with iterative refinement for more than two years. It is the first P4-based cloud gateway in the industry, of which a single cluster carries dozens of Tbps e-commerce traffic, successfully withstanding peak-hour traffic pressure from large online shopping festivals.

2 BACKGROUND AND MOTIVATION

2.1 Gateways for Cloud Networks

Public cloud networks. Different from the private cloud [29], which is operated for only a few selected users, the public cloud [26] packages its computing power and storage capability into sliced products, which are sold to a large number of customers (*i.e.*, tenants) for fine-grained resource sharing to achieve economics of scale [21]. Under such a multi-tenant architecture, cloud service buyers expect to build the entire network space to which they've subscribed with customized address segments, routing tables and ACL rules configured as if they are managing their own local networks. At the same time, the public cloud vendors expect as well that any tenant's (malicious) behavior will not affect the cloud resources used by others in order to guarantee the SLAs signed with their customers, even though the physical infrastructure is shared.

To satisfy the above requirements, VPCs [41] are built within the public cloud environment, providing logical resource isolation between multiple tenants. To create multiple VPCs transparent from each other, the public cloud network is no longer just a physical interconnection among network nodes for data transmission. At present, mainstream public cloud vendors rely on overlay network protocols (such as VXLAN [27]) to achieve network multiplexing and resource isolation, leaving the basic packet forwarding task to the underlay networks. Specifically, VXLAN is a framework for overlaying virtualized layer 2 networks over layer 3 networks and it uses a VXLAN network identifier (VNI) to identify a VXLAN segment. Only VMs within the same VXLAN segment (*i.e.*, sharing the same VNI) can directly communicate with each other. In this way, a VXLAN segment precisely implements a VPC for isolation.

The role of gateways in the cloud. The advantages of VPCs attract a large number of users, who establish connections to VPCs in the public cloud from their home, their office or any other place

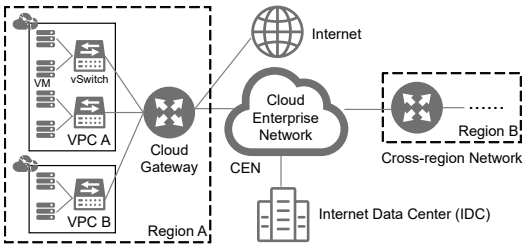


Figure 1: The cloud gateway is the central hub of diverse cloud traffic.

across the earth. Fig. 1 shows a high-level topology of the public cloud infrastructure of Alibaba Cloud. As shown in Fig. 1, Alibaba Cloud has multiple regions and each region contains a large number of VPCs. The VMs within the same VPC share the same VNI for network isolation. Multiple VMs are hosted on a physical server through a hypervisor. A vSwitch running within the hypervisor allows communication between VMs. In Fig. 1, IDC is the enterprise's local data center, from which employees can connect to their VPCs in the cloud. CEN is a dedicated leased line network between cloud regions and IDCs, providing high-speed IDC/cross-region communication. The *Cloud Gateway* plays an essential role in the public cloud infrastructure because it is the exact central hub of the east-west traffic (VM-VM), the south-north traffic (VM-Internet) and the IDC/cross-region traffic (VM-IDC/Cross-region) of the entire cloud. We elaborate on these traffic routes and the typical cloud services carried on them in Table 1. Take "IDC-VM" as an example. The traffic originated from the IDC will be forwarded hop-by-hop through the CEN, the Cloud Gateway, the vSwitch on the destination server until it finally reaches the destination VM. A typical service on this route is tenant's login to his VM from office.

Table 1: Typical cloud service examples on different traffic routes across the cloud gateway.

Traffic Routes	Cloud Service Examples
VM-VM (same VPC, different vSwitches)	Message passing for synchronization in distributed computing
VM-VM (different VPCs)	Communication between two cloud tenants within the same region
VM-Internet	Tenant crawls web pages from the Internet
Internet-VM	Tenant logs into his VM from home
VM-IDC	Tenant pulls the computation results from cloud to his office
IDC-VM	Tenant logs into his VM from his office
VM-Cross-region	Communication between one tenant from China and another tenant from USA

Packet forwarding at the cloud gateway. At the cloud traffic aggregation point, the responsibility of a cloud gateway is to (1) differentiate the arriving traffic according to the destination VM address and forward it to the right region/IDC/VPC containing the destination VM, and (2) find the physical server in the region/IDC/VPC where the destination VM is hosted according to the destination VM address. Once the physical server where the destination VM is hosted is located, the traffic will be sent to the server and the server will further deliver the traffic to the destination VM. According to the above forwarding process, the cloud gateway contains at least two important forwarding tables in its data path: (1) the VXLAN routing table, and (2) the VM-NC mapping table[1], as shown in Fig. 2. The VXLAN routing table finds

[1]NC is the abbreviation for Node Controller, which is the physical server hosting VMs and controlling their behaviors in Alibaba Cloud.

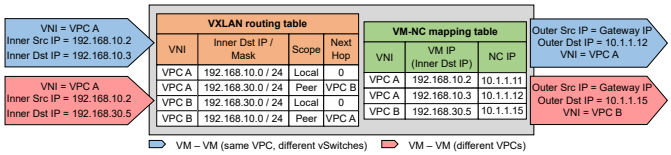


Figure 2: Packet forwarding at the cloud gateway.

the right region/IDC/VPC scope according to the inner DIP of the VXLAN-encapsulated packet. The VM-NC mapping table finds the exact physical server address where the destination VM is hosted according to the inner DIP of the VXLAN-encapsulated packet as well as the region/IDC/VPC scope obtained from the VXLAN routing table. Before leaving the gateway, the outer DIP of the packet will be modified with the server address obtained from the VM-NC mapping table for delivery of the packet to the destination VM.

In Fig. 2, we show the forwarding process of "VM-VM (same VPC, different vSwitches)" and "VM-VM (different VPCs)" in detail. "VM-VM (same VPC, different vSwitches)" means the two VMs are hosted on different physical servers but have the same VNI, while "VM-VM (different VPCs)" means the two VMs are on different physical servers and have different VNIs.

VM-VM (same VPC, different vSwitches). The arriving packet first queries the VXLAN routing table with VPC A (its VNI) and the IP address 192.168.10.3 (inner Dst IP of the VXLAN-encapsulated packet, destination VM address), and it hits the first entry of the VXLAN routing table, where the scope is "Local", indicating that the destination VM is also in VPC A. Then, it queries the VM-NC mapping table and hits the second entry, which shows the physical server IP of the destination VM is 10.1.1.12. Finally, the packet is forwarded with its outer Dst IP modified to 10.1.1.12.

VM-VM (different VPCs). The arriving packet first hits the second entry of the VXLAN routing table, where the scope is "Peer" and the "Next Hop" is VPC B, which means the destination VM is not in VPC A and the gateway should use VPC B as the VNI for further lookup of the VXLAN routing table until the scope becomes "Local". When querying the VXLAN routing table with VPC B and 192.168.30.5, it hits the third entry, and the scope is "Local", which means the destination is in VPC B, then it queries the VM-NC mapping table, finds its server IP in the third entry and gets forwarded.

2.2 Evolution of Software Gateways

Durable architecture for longer service time. As mentioned earlier, the cloud gateway is located at the traffic aggregation point, therefore, the stability of the system is critical to the reliability of cloud services since any gateway failure will explosively affect a massive number of users. The stability is reflected in two aspects: (1) the packet forwarding should be stable and bug-free (occasional crashes of the forwarding instances should be avoided), and (2) the system architecture should be durable for a longer service time (*e.g.*, 3-5 years) to save development expenses in a production environment, adapting to the rapid growth of the cloud size and the number of services (the architecture should be scalable to handle growing traffic and allow the iterative yet tractable upgrades to accommodate more services). Besides, the development should be agile to ensure rapid deployment of new services, especially given that this gateway was being developed during the early stages of Alibaba Cloud. By considering the above requirements,

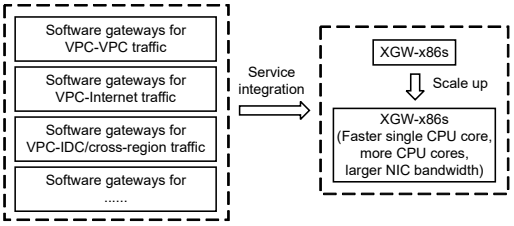


Figure 3: Evolution of the software gateway.

we designed XGW-x86, an x86-based software gateway to serve cloud traffic forwarding ("XGW" means eXtendable GateWay). We leveraged DPDK's kernel-bypass capability [9, 28] to accelerate the single-node performance (~1Mpps per CPU core) and used horizontal scaling [31] to further expand the packet processing capacity of the entire gateway cluster. Adding new services to XGW-x86 is easy due to its modular software design. The architecture has been tested by more than five years of production environment deployment with the following major evolution as shown in Fig. 3.

Service integration into one versatile gateway. For rapid deployment of new services, we built ad hoc clusters in the early stages of Alibaba Cloud, that is, one dedicated XGW-x86 cluster for one particular cloud service. In this way, the development, testing and online deployment of new services would not influence other already deployed services. As the traffic volume increased, some clusters expanded rapidly while other clusters were underutilized, both raised system CapEx. Furthermore, heterogeneous clusters needed different treatments (*i.e.*, we needed to assign dedicated engineers to different clusters for development and maintenance) while additional inter-cluster traffic handling was inevitable, both raised system OpEx. To reduce the CapEx and OpEx, when the cloud services became more stable, we determined to merge the heterogeneous software cloud gateways into one unified but versatile gateway, which extended the life of the architecture by reducing the CapEx and OpEx but without losing the scalability. After service integration, the monolithic system also became more stable with fewer bugs because we no longer needed to maintain separate, distinct codebases for different gateways.

Vertical scaling of single gateway processing capacity. The hardware speed keeps improving yearly. Taking advantage of this, XGW-x86 experienced a complete upgrade in its service time with more powerful CPUs (faster single-core performance, more CPU cores) and higher-bandwidth NICs. The vertical scaling of single-node capacity temporarily reduced the size of gateway clusters as well as the hardware acquisition and maintenance cost.

2.3 Limitations of Software Gateways

Although the durable XGW-x86 architecture has withstood the traffic processing load and cloud service changes in the past few years, with the persistent growth of traffic in Alibaba Cloud, it also accumulated many problems over time. Of course, other cloud vendors may well manage these problems still using the software-based solutions [16, 31]. Here, we try to provide observations and insights based on our own experiences in Alibaba Cloud.

High CapEx and OpEx of gateway clusters. Software gateways are clustered to handle the traffic in a cloud region. Suppose the traffic is 15Tbps [20, 30] and each gateway can maximally handle 100Gbps. Then, 15Tbps / 100Gbps = 150 gateways are required. In a production environment, we need to reserve enough processing headroom for service stability. If gateways are designed to forward at 50Gbps (*i.e.*, 50% water level), the gateway number will be doubled. Besides, considering the 1:1 backup for disaster tolerance, the number will be further doubled to 600! The unit cost of the software gateway is not cheap because each gateway contains 2/4 high-end Xeon processors [5] and 40/100GbE NICs. Suppose each gateway costs O($10K), the total cost will reach O($10M) for a single region. Software gateways also bring about considerable maintenance overhead for two reasons: (1) in the "scale-out" mode, each gateway shares the same forwarding tables and the controller needs to update all gateways simultaneously which is prone to latency and coherence issues (it takes more than ten minutes to install all the tables into one XGW-x86 gateway and it is time-consuming to update hundreds of gateways even though some degree of multi-threading is enabled at the controller), and (2) the large gateway number increases the difficulty of troubleshooting because more gateways introduce more potential failure sources.

Beyond the maximum next-hop limit of the load balancer. Cloud gateways are placed behind the load balancing switch/router which conducts ECMP flow-based forwarding [23] to distribute traffic equally to these gateways. However, commercial load balancers are generally limited to allowing fewer than 64 possible next-hops (*e.g.*, on Juniper Networks security devices, the maximum number of next-hop addresses in an ECMP set is 16 [3]), which constrains the size of the gateway cluster. To this end, hundreds of gateway nodes in a cloud region have to be partitioned into multiple smaller clusters behind different load balancers, which further increases the maintenance complexity.

CPU overload due to traffic bursts. Although we've already reserved sufficient safety margin in a real deployment, we still observe CPU core high utilization due to traffic bursts, especially during large online shopping festivals like "Double 11". In Fig. 4, we observe that a gateway's CPU core 1 is persistently overused during several days while the other CPU cores are all lightly loaded (for clarity, we only show the top-5 cores ranked by utilization out of 32). The CPU core overload will cause packet loss in a region from time to time as shown in Fig. 5 (note that the CPU core utilization shown in Fig. 4 is sampled in a coarse granularity and packet loss will occur when CPU core utilization reaches 100% even in a very short moment). In the worst time, the packet loss rate of the entire region reaches $\sim 10^{-5} - 10^{-4}$ (at Day 6). In a multi-tenant cloud, the CPU core overload will deteriorate the tenant's SLAs if his traffic is on that core. There are two possibilities for CPU core overload: (1) load imbalance among gateways, and (2) load imbalance among CPU cores. From Fig. 6, we find that the processing load is perfectly balanced among gateways during the same time period. That is, the load is unequally distributed among CPU cores.

Horizontal scaling is common, balancing among gateways is easy, balancing among CPU cores is not. We explore the root cause of the inter-core load imbalance. Fig. 7 shows the traffic proportion of top-N flows on the overused CPU core in historical CPU overload scenes. We find that in most cases, the top-1 and top-2 flows dominate the traffic. That is, traffic imbalance and a small number of heavy hitters contribute to the load imbalance among CPU cores. XGW-x86 follows the run-to-completion model, conducts flow-based hashing and distributes packets received from

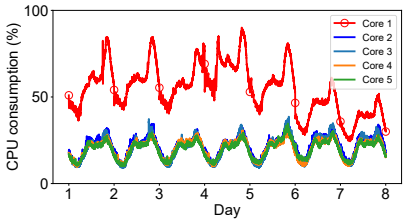


Figure 4: CPU overload in an XGW-x86 (we show top-5 cores out of 32).

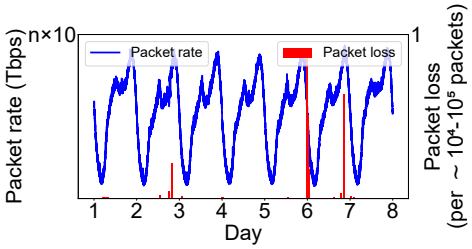


Figure 5: Traffic rate and packet loss rate of a region with XGW-x86s in a week.

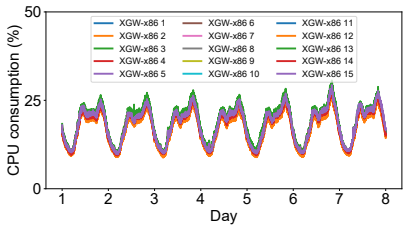


Figure 6: CPU consumption of XGW-x86s in the same region in a week.

a NIC to multiple RX queues via the RSS (receiver side scaling) technology [22], load balancing the packets among CPU cores. The flow-based hashing guarantees intra-flow in-order packet processing; however, it also causes potential CPU core overuse if multiple heavy-hitter flows are hashed into the same CPU core, even though the hashing algorithm itself is perfectly random. Sometimes, a single flow in Alibaba Cloud can even reach tens of Gbps, and the CPU overload problem directly affects the sale of bandwidth to important customers. Changing the run-to-completion model to a pipeline model may ameliorate the problem, but the pipeline model on x86 CPUs also has its own problems such as inter-core transfer performance penalty at the L3 cache [46]. Indeed, some network processors do have the ability to hash the packets of the same flow to multiple packet processing engines for packet-based load balancing and rely on dedicated sequence-preserving hardware for fast packet reordering [42]. However, the implementation of XGW-x86 based on general-purpose processors and DPDK cannot realize performant packet reordering due to the absence of dedicated hardware. Actually, load balancing among gateways/CPU cores is like randomly tossing balls into bins [33]. The smaller the bins, the higher probability of overflow. Although horizontal scaling is common wisdom in the cloud, it may not always work well.

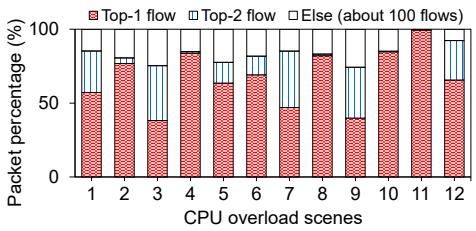


Figure 7: CPU overload caused by heavy-hitter flows.

Stagnant performance of the single CPU core. We have already described the root cause of the software gateway limitations, but a natural question to ask is: can it be smoothly fixed with future innovations in technology? We use history to predict the future, but the result is pessimistic. Fig. 8 shows Intel i7 CPU performance (single-core and multi-core) and ToR switch port speed from 2010 to 2020. The former roughly reflects the processing capability of software gateways and the latter partly reflects the traffic load faced by the cloud gateways. From 2010 to 2020, the port speed grew from 10GbE to 400GbE (40x), and the multi-core performance improvement was 4x; however, the single-core improvement was only 2.5x. The Internet traffic growth is far beyond Moore's law [34] while Moore's law also goes beyond the performance improvement of a single CPU core. Based on these results, XGW-x86 will persistently be challenged by more and more heavy hitters.

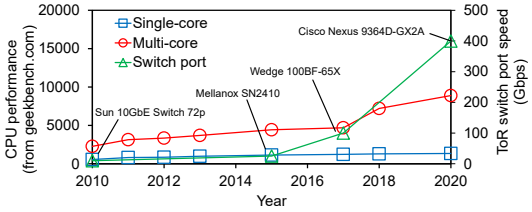


Figure 8: CPU performance (single-core and multi-core) and ToR switch port speed from 2010 to 2020.

3 HARDWARE GATEWAY AND CHALLENGES

3.1 Hardware Options

The exposed problems of XGW-x86 push us to find a more robust solution to protect the cloud gateway from traffic floods. In late 2017, we naturally turned to hardware and thoroughly investigated the following three candidates considering speed, cost, programmability and power dissipation.

Fixed-function ASICs. Fixed-function ASICs such as Broadcom Tomahawk 2/3 [1] deliver more than 10Tbps switching throughput with constrained power consumption and affordable cost. These chips are originally made for high-speed switches, due to the lack of full programmability, however, they are not the best fit for our versatile cloud gateways. First, Alibaba Cloud deploys many proprietary protocols (such as Vtrace [17]) which need extra support from ASIC vendors. Second, fixed-function ASICs cannot well adapt to the persistent changes of cloud services. The time-to-market cycle of ASICs is too long for the service-driven cloud networks.

FPGAs. FPGAs are fully programmable but have a steep learning curve. The switching performance of FPGAs is generally below 400Gbps due to their relatively low frequencies, which make them more suitable for building smartNICs [19] at the end hosts rather than cloud gateways at the central hub. Recently, some proposals try to push FPGA's processing flexibility into the network core [32]. Besides, FPGAs are more expensive than fixed-function ASICs and more power-hungry for equivalent performance.

Programmable ASICs. Programmable switching ASICs have both advantages of fixed-function ASICs and FPGAs that it allows programmers to flexibly customize their forwarding logic and to quickly adapt to cloud service changes while retaining the high performance and low power consumption [10, 11]. Besides, programmable switching ASICs have an affordable price. According to our knowledge, the Tofino-based switch has roughly the same unit price as XGW-x86. That is to say, we can replace many XGW-x86 gateways with a single Tofino-based switch that can provide equal forwarding performance but at a much-reduced cost. In late 2017, we had very few options for implementing the cloud gateway with

Table 2: Table size and table occupancy in the Tofino chip.

Table name	Key	Match type	IP type	Key length (bit)	Match SRAM	TCAM
VXLAN routing table	VNI, Inner Dst IP	LPM	IPv4	24 + 32		311%
			IPv6	24 + 128		622%
VM-NC mapping table	VNI, Inner Dst IP	EXACT	IPv4	24 + 32	58%	
			IPv6	24 + 128	233%	
Sum (75% IPv4, 25% IPv6)					102%	388.75%

programmable ASICs so we choose the Tofino at that time. But in 2021, numerous chip vendors have announced their programmable switching ASICs (*e.g.*, Broadcom's Trident 4 [4], Cisco's Silicon One [2]) and we believe our high-level design paradigm and the experiences with Tofino can be extended to other chips.

3.2 Programmable Switching ASICs

Fig. 9 shows the Tofino's pipeline-based forwarding architecture. The Tofino has four independent/isolated pipelines for packet processing. On each pipeline, arriving packets will pass through the Parser/Match-Action Unit/Deparser of the Ingress Pipe and those of the Egress Pipe in turn. Each pipeline has SRAM and TCAM memory resources which are equally distributed to multiple stages of that pipeline. Specifically, each stage has its own SRAM and TCAM, and cannot access the memory resources of other stages even in the same pipeline (memories in other pipelines are also inaccessible). The on-chip memories are very limited (O(10MB)) [24], and the amount of TCAM is much less than that of SRAM. The intermediate result from the table lookup at each stage is kept in metadata, which is shared within the entire Ingress or Egress Pipe but cannot directly transfer between them.

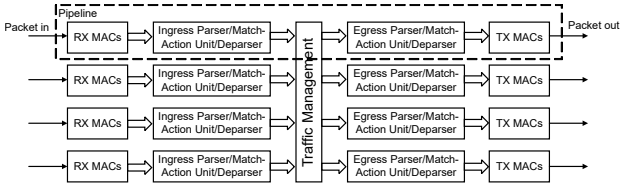


Figure 9: Tofino's packet forwarding architecture.

3.3 Technical Challenges

Larger table entry numbers with longer table entries. The VXLAN routing table and VM-NC mapping table cover the majority of cloud traffic. In the public cloud, the growth of tenants/VPCs, as well as the interconnection between them, contributes to the expansion of the VXLAN routing table. The growth of VMs allocated by tenants contributes to the expansion of the VM-NC mapping table (some top customers can purchase millions of VMs even in a single VPC). According to our statistics, a large cloud region in Alibaba Cloud can have O(1M) VPCs and O(1M) VMs, leading to a very large VXLAN table and VM-NC table. Except for the table entry number, both the VXLAN table and VM-NC table have longer table entries compared with that of L2/L3 devices. Because in addition to storing the destination IP addresses, the VXLAN table and VM-NC table also need to store the 24-bit VNI field.

Limited total on-chip memory capacity. Storing the O(1M) VXLAN routing table and O(1M) VM-NC mapping table is easy for the XGW-x86 due to the large-capacity external DRAM; however, the same is difficult for the above-mentioned Tofino chip with its smaller O(10MB) on-chip memories. In addition to limited memory resources, the ratio of SRAM/TCAM is also fixed when the silicon is taped out, which cannot adapt to the changing cloud service requirements. Besides, the TCAM is significantly more scarce due to its expensive price. However, the LPM lookup requirement is not a small percentage compared with the exact match for the cloud gateway traffic. Table 2 shows that if we straightforwardly put the two major tables into the Tofino chip, the memories required by these tables will be much more than what is available.

Handling diverse cloud services. Besides the VXLAN routing table and VM-NC mapping table, the gateway also needs to carry other tables for diverse cloud services. Some tables are stateful and ultra-large in table entries, such as SNAT with O(100M) entries. Some tables are volatile, such as temporary tables allocated for on-demand traffic load balancing that is activated only during online shopping festivals. Some tables are QoS-related and installed based on the SLAs signed with customers, such as meter, counter, ACL tables. Given the limited on-chip memories, how to arrange these heterogeneous tables in the Tofino needs careful consideration.

Processing yearly growing IPv6 traffic. In addition to handling IPv4 traffic, since more and more customers are transferring their business to IPv6 [14], the memory design for the cloud gateway faces even more challenges. First of all, IPv6 has much longer table entries, which makes the overall memory consumption much higher. Second, as the demand for IPv6 continues to grow, we need to consider dynamically allocating the on-chip memories to meet the need of growing IPv6 table entries. The dynamic memory allocation is prone to producing memory fragments, resulting in low utilization of the originally insufficient on-chip memories. Therefore, how to allocate memories on-demand to efficiently satisfy both IPv4 and IPv6 becomes another major challenge.

Table placement on noncontinuous memory space. The Tofino has multiple pipelines and each pipeline has multiple stages. Each stage has its own local memory resources that cannot be accessed from other stages. In other words, the memory space is not flat and continuous on the programmable ASICs, totally different from that of the x86 architecture, where the virtual memory mechanism guarantees an entirely continuous address space. Fortunately, large tables across stages within the same pipeline can be automatically handled by the Tofino's compiler with table splitting and mapping to multiple stages. However, the compiler does not address table placement across pipelines. Besides, as mentioned before, metadata transferring between Ingress Pipe and Egress Pipe is not allowed in the Tofino architecture as another architectural constraint for the programmer. To summarize, we should be thoroughly aware of the architectural constraints to map the logical tables as well as their relationship to the physical memory layout.

4 DESIGN AND IMPLEMENTATION

4.1 Design Overview

In order to successfully handle cloud-scale (high throughput), multi-tenant (huge VXLAN routing table and VM-NC mapping table) and multi-service (diverse forwarding tables) traffic, we design *Sailfish*, a hardware/software integrated cloud gateway system. The overall

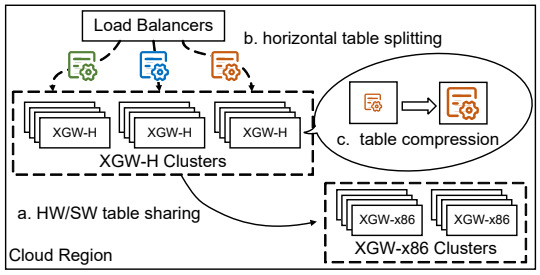


Figure 10: Architecture of Sailfish.

architecture and high-level designs are shown in Fig. 10. We leverage a multi-pronged approach to solve the problem of insufficient on-chip memories from three perspectives.

Hardware and software co-design. *Sailfish* consists of both Tofino-based hardware gateways (XGW-H) and x86-based software gateways (XGW-x86). XGW-H is ultra-fast but with limited memories and constrained memory layout while XGW-x86 has huge memory space with full programmability but limited performance. By analyzing traffic characteristics of different cloud services, we place XGW-H in front of XGW-x86 in a region. We then leverage XGW-H to store a few key tables frequently hit by the majority of traffic, and use XGW-x86 as a complement to hold the remaining volatile tables hit by a small portion of traffic and huge stateful tables that cannot be compressed into XGW-H. (a in Fig. 10).

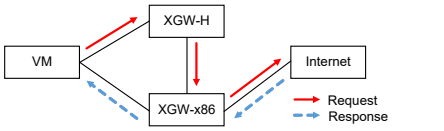
Table splitting among XGW-H clusters. *Sailfish* conducts horizontal table splitting among XGW-H clusters in a region to further reduce the number of table entries stored in each XGW-H. Here, "horizontal splitting" means each XGW-H stores all the forwarding tables but only a portion of entries from each table. Multiple XGW-H clusters cover all the entries in a region, while a single cluster is only responsible for entries of some tenants. Within a cluster, multiple XGW-H devices maintain the same table entries, share the traffic load and backup for each other. (b in Fig. 10).

Single-node table compression. *Sailfish* conducts table compression on each XGW-H in order to greedily fit more table entries into the single node. These optimizations include pipeline folding, table splitting between pipelines, table mapping across pipelines, memory resource pooling, TCAM conservation and table entry compression. The single-node table compression increases the number of entries carried in one cluster, thus reducing the number of necessary clusters, CapEx and OpEx. (c in Fig. 10).

4.2 Hardware and Software Co-Design

Due to the rapid growth of cloud traffic, it is difficult to carry all the traffic with XGW-x86. The emergence of programmable switching ASICs allows hardware gateways to handle a variety of cloud services with high performance. However, the on-chip memories are too limited for hardware gateways to store all the table entries. Therefore, the combination of software and hardware is a feasible direction in the evolution of cloud gateways. Through data mining of real cloud traffic, we find that the traffic exactly follows the "80/20 rule". For example, in a typical cloud region, 5% of the table entries carry 95% of the traffic, and the remaining 95% of the entries only carry 5% of the traffic. Based on this observation, we formulate the principles of hardware and software cooperation.

- XGW-H is the default gateway which is placed in front of XGW-x86 to absorb the majority of traffic.



SNAT session table (in XGW-x86)	
Session Five-tuple (Inner Src/Dst IP, Inner L4 protocol, Inner Src/Dst port)	Public Network IP / Src port
Session 1 for VM 1 (Inner Src IP = VM 1 IP)	XGW-x86 IP 1 / Port 1
Session 2 for VM 1	XGW-x86 IP 2 / Port 2
.....	.....
Session *n* for VM *k*	XGW-x86 IP 3 / Port 3

Figure 11: XGW-H and XGW-x86 cooperation for stateful SNAT traffic handling.

- XGW-H stores a few key tables frequently hit by the majority of traffic, ensuring the reliability and stability of basic cloud services. Besides, XGW-H is responsible for guiding the remaining traffic to XGW-x86.
- XGW-x86 maintains a large number of volatile tables, hit by a small portion of traffic. It also stores large-sized stateful tables that cannot be easily compressed into XGW-H. The full programmability of the continuous memory space in XGW-x86 eases frequent table updates from long-tail services.
- Generally, unstable newborn services with less traffic are carried by XGW-x86, while mature services with heavy and stable traffic are turned over to XGW-H. In this way, the failure of the unstable services with higher probabilities will not affect the traffic forwarding at XGW-H. All of the table-sharing decisions are predetermined by the central controller.
- Considering the huge difference in performance, rate limiting is necessary at XGW-H before forwarding the traffic to XGW-x86 for overload protection.

Fig. 11 shows an example of cooperation between XGW-H and XGW-x86 for stateful SNAT traffic handling. Some customers own a large number of VMs but only a few public IPs. In order for all the VMs to access the public network, SNAT is needed in the cloud gateway. SNAT maps the 5-tuple to the public network IP and port. Hence, the number of entries in the SNAT table is decided by the number of sessions, which is much larger than the number of VMs. The entry number of the SNAT table can reach O(100M), while that of the VM-NC table is only O(1M). The SNAT table is too large to fit in XGW-H, and the use frequency of each entry is also not very high. Besides, the mapping between the session and the public IP/Src port is volatile. So we put the SNAT table in XGW-x86.

As shown by the red arrow, the VM visits the public network through SNAT. The outgoing packet first reaches XGW-H. According to a special VNI tag carried in the packet, XGW-H determines that the packet wants to visit the public network and requires SNAT. Then, it forwards the packet to XGW-x86. XGW-x86 finds the corresponding public IP and port according to its 5-tuple and then replaces its inner Src IP and inner Src port, removes the VXLAN tunnel and forwards the packet. The blue arrow is the response from the public network. The public network sends a response packet to the VM according to the Src IP of the request packet, so the response packet will directly reach XGW-x86, and then XGW-x86 adds a VXLAN tunnel and sends it to the VM.

After table sharing between XGW-x86 and XGW-H, for a typical cloud region with tens of Tbps of traffic, considering both safety margin and disaster tolerance, we sharply reduce the hardware

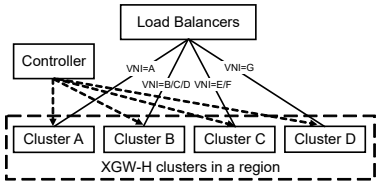


Figure 12: Table splitting among XGW-H clusters.

cost from hundreds of XGW-x86s to ten XGW-Hs for major traffic processing and four XGW-x86s for fallback traffic processing.

4.3 Table Splitting Among XGW-H Clusters

As shown in Fig. 12, in a region, we split the table entries horizontally based on VNI into different XGW-H clusters. Each cluster maintains all the tables and a portion of entries from each table. The table partition and entry distribution are managed by the central controller. At the data plane, traffic is distributed according to the VNI via a load balancer in front of the XGW-H clusters. There are four major benefits of horizontal table splitting compared with vertical table splitting or flow-based hashing.

- Good scalability. Straightforwardly maintaining all the table entries in each cluster is apparently not scalable due to the limited on-chip memories of XGW-H. But what if we conduct vertical table splitting, that is, splitting different tables of all entries into different clusters? In fact, the merit of horizontal splitting is that when new tenants are added into the cloud, we only need to insert new table entries into one cluster or allocate a new cluster if the original cluster is out of memory. But vertical table splitting cannot achieve this.
- Fault isolation. In a real deployment, software and hardware bugs caused by rush development or misconfiguration are inevitable for such a complex system. Horizontal splitting can effectively constrain the influence scope of some faulty entries within one cluster, thus the failure of one cluster will not influence others in the same region. Otherwise, the influence scope will extend to the entire region, even if most clusters have nothing to do with those faulty entries.
- Tractable traffic load balancing. Indeed, for load balancing, we can also use flow-based hashing to distribute traffic to multiple clusters. However, load balancing via hashing is somewhat uncontrollable. By contrast, horizontal splitting can precisely manage the traffic load on a particular cluster simply by adding or deleting the corresponding entries.
- Reduced maintenance complexity. Horizontal table splitting maintains the same tables and lowers the number of table entries in the cluster, therefore, reducing the table management complexity as well as the disaster recovery time.

4.4 Single-Node Table Compression

After table splitting among clusters, each cluster only needs to carry part of the table entries. But it does not mean that we can split infinitely and use an unlimited number of clusters to hold the entries. There are two reasons: (1) table splitting has a limit since the VPC is the smallest split granularity, however, some VPCs (*e.g.*, top customers) contain millions of entries that challenge the capacity of a single cluster; and (2) we need to consider the CapEx and OpEx to build clusters. Hence, we further conduct single-node optimizations to expand XGW-H's capacity to hold more entries.

Pipeline folding. XGW-H can originally reach 6.4Tbps forwarding throughput if the four pipelines contain the same forwarding

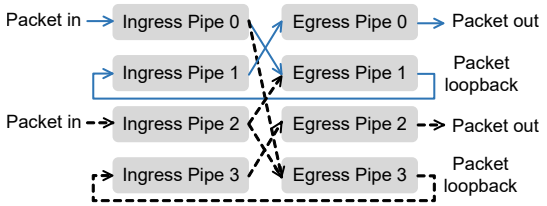


Figure 13: Pipeline folding for storing more table entries.

tables and work in parallel. The throughput can further be increased by adding more gateways in a cluster. That is, for XGW-H, throughput is sufficient and easy to extend while memories are in real shortage. Accordingly, we propose "pipeline folding" as a trade-off to sacrifice the throughput by halving the working pipelines for doubled memory capacity in each "folded" pipeline. As shown in Fig. 13, we use Ingress Pipe 0/2 as the packet entry point. Then, packets enter Egress Pipe 1/3, but the ports of those pipes are set to Loopback mode, so packets will enter Ingress Pipe 1/3 through the same ports after leaving Egress Pipe 1/3. Finally, packets will be sent out through Egress Pipe 0/2 (Ingress Pipe 1 goes to Egress Pipe 0, Ingress Pipe 3 goes to Egress Pipe 2). In addition to halving the forwarding throughput, the design will also double the forwarding latency. However, adding O(1μs) to the end-to-end latency is imperceptible for the end-users. But the doubled memory capacity for holding more table entries is indeed a huge gain. Considering the architectural constraints of the Tofino, we summarize the principles of distributing tables in the four pipelines as follows.

- Tables should be placed in Ingress Pipe 0/2, Egress Pipe 1/3, Ingress Pipe 1/3, and Egress Pipe 0/2, following the table lookup order to match the packet flow in pipeline folding.
- Metadata cannot be directly transferred between Ingress Pipe and Egress Pipe. If metadata transfer across different pipes is indeed required, we have to append metadata to the packet, which is called bridging. Bridging will increase the packet length thus affect the forwarding throughput. With pipeline folding, the number of possible bridges increases from 1 to 3. Hence, for tables that need to share the same metadata, we recommend placing them in the same pipe to minimize the number of possible bridges as well as the throughput loss.
- Tables should be evenly distributed in different pipelines, ensuring that each pipeline has enough room for table expansion in the future, making the system extensible without frequently re-mapping the existing tables.

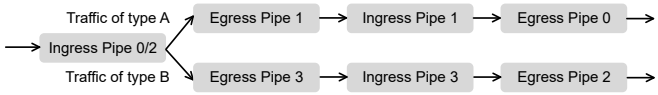


Figure 14: Traffic splitting between pipelines.

Table splitting between pipelines. When packets leave Ingress Pipe 0/2, they will enter Egress Pipe 1/3. Originally, Egress Pipe 1 and Egress Pipe 3 contain the same table entries. To reduce the table redundancy, we can store some entries in Egress Pipe 1 and the other entries in Egress Pipe 3, using some hash functions to divide the traffic into different pipes (Fig. 14). The horizontal table splitting design described here is very similar to that in §4.3. The difference is that we split tables between pipelines on the Tofino chip rather than split tables among clusters in a region. The general principle is to split the entries as equally as possible to ensure

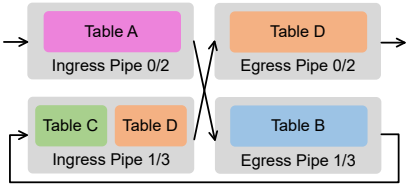


Figure 15: Mapping large tables across pipelines.

that traffic carried by each pipeline is balanced. For example, we can split entries according to the parity of VNI or inner Dst IP.

Mapping large tables across pipelines. After pipeline folding, the tables in Pipeline 0 and Pipeline 2 are the same, and the tables in Pipeline 1 and Pipeline 3 are the same. The tables in Pipeline 0/2 and Pipeline 1/3 together provide the complete forwarding logic. Since the memories of each pipeline are isolated, there may be an imbalanced memory occupation between Pipeline 0/2 and Pipeline 1/3. As shown in Fig. 15, the table lookup order is Table A-B-C-D. We have placed Table A in Ingress Pipe 0/2, Table B in Egress Pipe 1/3, and Table C in Ingress Pipe 1/3. At this time, there is still some free space in Pipeline 1/3, but it is not enough for holding the entire Table D. While there is a large free space in Pipeline 0/2. We place part of Table D in Ingress Pipe 1/3 to make full use of Pipeline 1/3, while we put the rest in Egress Pipe 0/2, consuming the remaining memory of Pipeline 0/2 to store the rest of Table D. By mapping large tables across pipelines, we break the isolation of resources between different pipelines and make table placement more free.

IPv4/IPv6 table pooling. It is feasible to allocate dedicated IPv6 tables alongside IPv4 tables to allow them to be updated independently from each other. In this way, the expansion of one table will not affect the service provided by another table. However, since the traffic ratio of IPv4/IPv6 is changing constantly, separate tables may cause memory waste or insufficient memory due to the inefficient memory occupation. Our strategy is to pool IPv4 and IPv6 memory resources. For any table with IP as its key, both IPv4 and IPv6 are supported, ensuring that the ratio of IPv4/IPv6 can be adjusted arbitrarily. Specifically, the IPv4 key can be expanded to a 128-bit to align with the IPv6 key in the same table; or the IPv6 key can be compressed to a 32-bit to align with the IPv4 key. For the VXLAN routing table, we choose the first strategy for ease of conducting the LPM search; for the VM-NC mapping table, we choose the second strategy because it just requires the exact match. The table pooling strategies can also be applied to other cloud services.

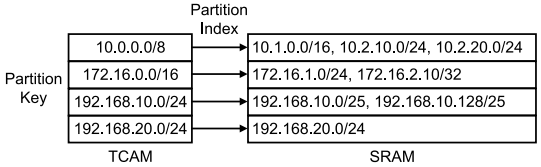


Figure 16: Forwarding table partition with ALPM.

TCAM conservation for large FIBs. Accommodating the entire VXLAN routing table with TCAM can ensure search efficiency. But the TCAM resource on the Tofino is very limited. Besides, the memory ratio of TCAM/SRAM is a constant for a chip, but the table lookup requirements of the LPM or the exact match change over time. To address this, we implement algorithmic LPM (ALPM) [40] to flexibly reduce the TCAM usage at the cost of slightly reduced lookup efficiency and more SRAM usage. As shown in Fig. 16, the entire routing table is partitioned into two levels with the first level stored in TCAM, indexing the second level stored in SRAM. The tradeoff between TCAM occupancy and table lookup efficiency can be made by adjusting the depth of the first level.

Table 3: Memory occupancy after optimizations.

Table name	Match SRAM	TCAM
VXLAN routing table	18%	11%
VM-NC mapping table	18%	
Sum	36%	11%

Compressing longer table entries. If the key of table entries is too long, we try to compress it to a shorter hash digest to save memory space. For example, when building the VM-NC table with the IP dual-stack, we conduct IPv4/IPv6 table pooling and compress the IPv6 key to a 32-bit for alignment. The compression from 128-bit to 32-bit for IPv4/IPv6 table pooling will cause two kinds of conflicts. The first is between compressed IPv6 and original IPv4, which can easily be distinguished by using an additional label in the table entry. The second is between two compressed IPv6 keys, which can be resolved with an extra small table to hold the conflicting entries containing the complete 128-bit key. When conducting IPv6 lookup in the VM-NC table, we will first search the conflicting table with the 128-bit key, and then the IPv4/IPv6 table with the 32-bit compressed key if there is no item in the conflicting table. According to our experience, the 128-to-32 compression by hashing will generate very limited conflicts, and thus, the table dedicated to conflict resolution will not consume much memory.

Memory usage after optimizations. After the above optimizations, the two major tables, the VXLAN routing table and the VM-NC mapping table, can finally be fit in the Tofino chip (as shown in Table 3). Since we have conducted IPv4/IPv6 table pooling, the memory occupancy will not further change with the traffic ratio of IPv4/IPv6. Compared with the case-by-case results in Table 2, in the 100% IPv4 scenario, the use of SRAM is reduced by 38%, and the use of TCAM is reduced by 96%; in the 75% IPv4 and 25% IPv6 scenario, the use of SRAM is reduced by 65%, and the use of TCAM is reduced by 97%; in the 100% IPv6 scenario, the SRAM usage is reduced by 85%, and the TCAM usage is reduced by 98%.

5 EVALUATION

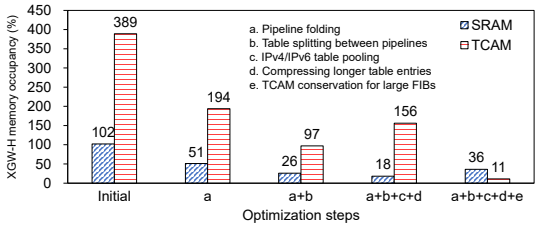
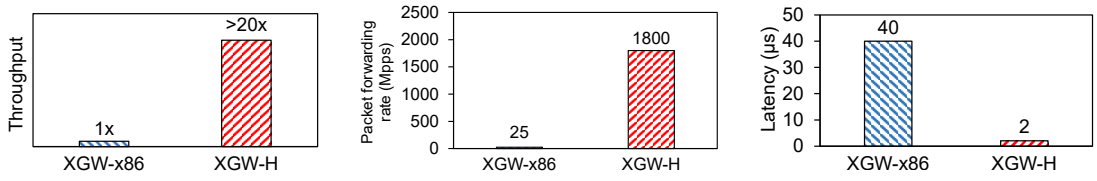


Figure 17: Memory usage after step-by-step compression.

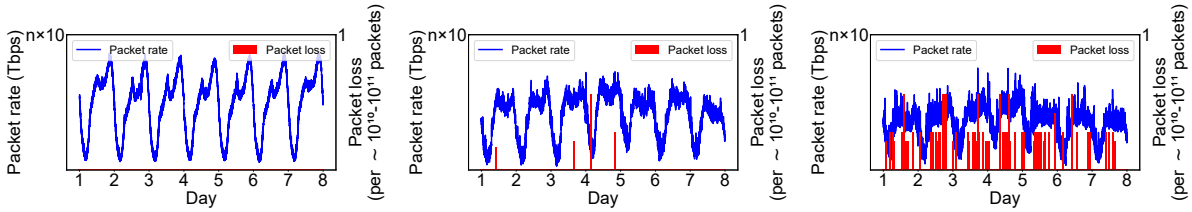
5.1 XGW-H Performance

XGW-H is developed on Tofino 6.4T with thousands of lines of P4-16. We evaluate the single-node performance of XGW-H in terms of step-by-step memory optimizations, overall memory consumption as well as forwarding performance, which is compared with XGW-x86. Then, we evaluate Sailfish's performance in a real deployment, focusing on the long-term packet loss in large cloud regions, the traffic load distribution between pipelines, traffic sharing between hardware and software, and table update frequencies.



(a) Throughput (b) Packet forwarding rate (c) Latency

Figure 18: XGW-H's forwarding performance.



(a) Region A (b) Region B (c) Region C

Figure 19: Sailfish's performance in three large cloud regions during a one-week online shopping festival.

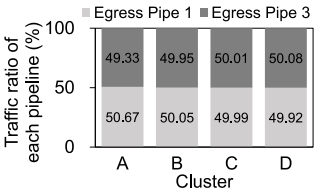


Figure 20: Balanced traffic distribution between pipelines (view of clusters).

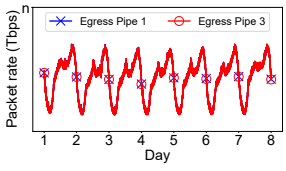


Figure 21: Balanced traffic distribution between pipelines (view of time).

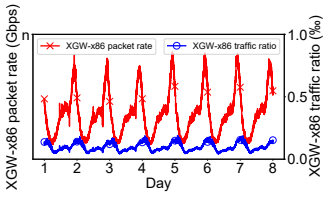


Figure 22: Minority of traffic hits XGW-x86 which contains majority of forwarding tables.

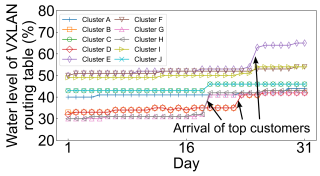


Figure 23: Regular updates and sudden updates of the VXLAN routing table.

Step-by-step table compression. Fig. 17 shows the memory occupancy of the VXLAN routing table and the VM-NC mapping table after step-by-step table compression. Before any optimization, the SRAM and TCAM occupancy reaches 102% and 389% (recall Table 2). After pipeline folding, the memory occupancy of both SRAM and TCAM is reduced by half at the cost of sacrificed throughput. After table splitting between pipelines, the memory occupancy in each pipeline is further reduced by half. In IPv4/IPv6 table pooling, we extend the IPv4 key to a 128-bit for alignment in order to conduct the LPM lookup, and therefore, the TCAM occupancy grows. Meanwhile, we also compress the IPv6 key to a 32-bit for alignment in the VM-NC mapping table with the exact match. The result is decreased SRAM occupancy. Finally, after conducting ALPM for TCAM conservation, the TCAM occupancy is greatly reduced to 11% while the SRAM occupancy is doubled to 36%.

Table 4: Overall memory resource consumption.

Pipeline	Match SRAM	TCAM
Pipeline 0/2	70%	41%
Pipeline 1/3	68%	22%
Sum	69%	32%

Overall memory consumption. Table 4 shows the overall memory resource consumption of XGW-H considering all the actual tables. Pipeline 0/2 and Pipeline 1/3 occupy 69% of SRAM and 32% of TCAM. Our design effectively solves the memory shortage problem of storing millions of table entries in programmable ASIC chips, and there is still room for adding future table entries.

Forwarding performance. Fig. 18 shows the forwarding performance comparison between XGW-H and XGW-x86 in terms of throughput, packet rate and latency. Under roughly the same unit price, the throughput of XGW-H is more than 20x that of XGW-x86 (Fig. 18(a)). When the port bandwidth is full, we pressure test the packet forwarding rate of XGW-H and XGW-x86. As shown in Fig. 18(b), the packet rate of XGW-H is 72x that of XGW-x86. Even for very complex cloud service logic, XGW-H can still reach line rate with packets smaller than 256B, while XGW-x86 reaches line rate with packets larger than 512B. We test the forwarding latency of XGH-H and XGW-x86 without background traffic. As shown in Fig. 18(c), although XGW-H uses pipeline folding to make the packet pass through one more pipeline, the average latency is still only 2μs, which is 95% lower than that of XGW-x86. For XGW-H, the latency varies from 2.173μs to 2.303μs for 128B-1024B IPv4 traffic and from 2.177μs to 2.306μs for 128B-1024B IPv6 traffic.

5.2 Sailfish Performance

Packet drop rate. Fig. 19 shows the performance of Sailfish in three large cloud regions during a large online shopping festival. The peak traffic of the three regions are dozens of Tbps. It shows that Sailfish delivers very stable performance in a production environment with only minor packet drop rates from 10^{-11} to 10^{-10}, which are six orders of magnitude lower than that of XGW-x86 (recall Fig. 5). The ultra-low packet drop rates mainly benefit from the large safety margin provided by the Tofino's high-capacity pipelines.

Traffic load distribution between pipelines. We horizontally split traffic between pipelines to reduce the table entries in each pipeline by half. The split method can be either hashing or leveraging service characteristics with historical data mining. As shown

in Fig. 20 and Fig. 21, the traffic can be partitioned in a good balance between pipelines in both cluster and time dimensions. Unlike CPU overload caused by heavy-hitter flows, it can hardly cause extreme unevenness between pipelines because the pipeline number is small and each pipeline has a huge processing capability.

Traffic sharing between XGW-H and XGW-x86. Fig. 22 illustrates the volume and proportion of traffic carried by XGW-x86 in Sailfish after conducting the table-sharing strategy. It shows that in the production environment, XGW-x86 carries a very small portion of cloud traffic ($< 0.2‰$). That is, the majority of traffic hits the minority of tables, which is absorbed by XGW-H with ultra-low latency. The remaining few Gbps of traffic can safely be handled by XGW-x86 without causing any CPU core overload problem.

Table update frequencies. Fig. 23 shows the changes of entry numbers in the VXLAN routing tables of different clusters during a month. For most of the time, the table is updated very slowly with sudden increases of table entries occurring infrequently. Since regular table updates occur at a relatively low frequency, they can be handled without challenges. The sudden increases are mainly ascribed to the arrival of top customers who purchase a large number of VMs or conduct a batch of route updates all at once. Although the sudden changes will potentially affect the stability of our system (*e.g.*, causing cluster expansion suddenly), we will be informed by the top customers and will be aware of them ahead of time in a real deployment. As long as we can prepare all the updated table entries before the arrival of the top customers' traffic, we can still handle this special case without any risk.

6 EXPERIENCES

6.1 Deployment Experiences

Cluster construction. In a production deployment, the number of allocated cloud gateways depends on customers' service requirements. Before putting gateways online, all table entries will be downloaded first from the central controller to all the gateways. Due to the large number of entries and gateways, table entry inconsistency between the controller and the gateways may occur during table population due to software/hardware bugs, misconfiguration or insufficient gateway memory [43]. Therefore, periodic consistency checks are needed. Then, we will deploy probe generators to produce diverse probe packets covering as many test scenarios as possible. Finally, we will modify the routes in the upstream devices to admit user traffic into the gateway clusters. If the user traffic is too heavy, we will admit the traffic incrementally.

Cluster management. During the runtime of gateway clusters, we periodically monitor the table water level, traffic rate and packet loss rate. We have to deploy new clusters in two cases: (1) the table size exceeds the available memory, and (2) the traffic volume exceeds the available processing power. Generally, the table entry growth is easy to predict with linear growth most of the time and sudden growth informed by top customers ahead of time. By contrast, traffic growth is difficult to predict. We have to rely on the safety margin provided by XGW-H to accommodate traffic bursts and deploy new clusters if the average traffic volume gets too high. In practice, we will reserve a safe water level for tables in XGW-H and XGW-x86 to allow tenants to add new entries. When the water level is close to the safe threshold, we will temporarily close the sale of the cluster's resources and consider putting new users in another cluster or constructing a new cluster. To ensure stable forwarding performance, each port will also reserve a safe water level to monitor the traffic rate and packet loss rate. If the packet loss rate is close to the safe threshold, the controller will be alerted to take further actions. At online shopping festivals with huge traffic volumes, we will deliberately raise the safe water level to further increase the gateway's allowable throughput by reducing the number of alerts sent to the controller. Other system metrics such as CPU, memory and disk utilization will also be monitored.

Disaster recovery. Disaster recovery is designed at different levels including cluster, node and port. At the cluster level, all the gateway clusters strictly follow 1:1 backup. The backup clusters are hot standby with the same configuration as the main clusters. The health status of the main clusters is monitored in real time and any anomaly will alert the controller to modify the routes in the upstream devices for traffic reroute to the backup clusters. At the node level, when some gateway reports hardware failures or its safe water levels are approached, the gateway will be put offline and the other gateways in the same cluster will share the traffic load of that gateway. If the gateways in the same cluster are all busy, we will resort to globally reserved cold standby gateways. At the port level, when a port suffers abnormal jitters or persistent packet loss, it will be isolated and the traffic will be migrated to other ports by modifying the routes in the upstream devices.

6.2 Lessons Learned

Is Sailfish ready to serve outside Alibaba? We have invested considerable man-months into the conception and realization of Sailfish to fit cloud-scale forwarding tables into the Tofino's limited on-chip memories. In Alibaba Cloud, we are driven by the violent traffic growth and have no choice but to turn to hardware. To date, Sailfish's performance has not failed us, so it is worth it. But from the perspective of small and medium-sized cloud vendors, in the short term, XGW-x86 may be good enough to solve their problems. Even if programmable switches are used, they may not face the problem of insufficient memory resources. But in the long term, as the cloud business is growing rapidly while CPU performance improvement is slowing down, we argue that they will face these problems sooner or later. Hence, we believe Sailfish is not trivial.

What about Sailfish's long-term viability? Sailfish's long-term viability is mainly affected by the trend towards more or less SRAM/TCAM consumed per Gbps. Generally, the memory consumption (*i.e.*, table size) is decided by the number of VMs while the traffic rate is decided by tenants' service requirements. One extreme case is that the number of VMs increases rapidly while the traffic rate growth stalls. In this case, Sailfish will continuously be challenged by the memory shortage problem under the rapid growth of table entries. According to our observation, however, the traffic rate per VM has been increasing persistently over the years in Alibaba Cloud, which also means the SRAM/TCAM consumed per Gbps has been decreasing. If such a trend is not going to be reversed in the future, Sailfish should have long-term viability.

How difficult is the development? P4 is a domain-specific language [10]. Compared with the XGW-x86 developed with C, the development process of XGW-H is even slightly simpler, thus the agile deployment of new services can also be guaranteed. However, the developers still need to understand the Tofino's pipeline

model and memory constraints for further optimizations. Besides, as a recent innovation, programmable switching ASICs have insufficient toolchains, such as test tools for code coverage. We needed to write lots of test cases and manually reviewed the results to verify the integrity of our functions. We believe a complete toolchain can promote the development efficiency of programmable ASICs.

How difficult is it to manage the heterogeneous clusters? Sailfish is a heterogeneous gateway system containing both XGW-Hs and XGW-x86s. Generally, heterogeneous systems are more complex and hard to manage compared with homogeneous systems. The complexity is reflected in different node forms as well as the consistency and coordination between heterogeneous nodes. In Sailfish, we try to manage the complexity via building a unified abstraction layer over the heterogeneous nodes. In this way, the network operators or the controller are unaware of the low-level difference between XGW-H and XGW-x86. The past experiences of managing the homogeneous XGW-x86s can be inherited.

Building stable gateways for a production environment. For cloud gateways at the central hub of the entire cloud, stability and performance are equally important. Failures in cloud gateways will affect a massive number of users and their services. For example, in our design, forwarding tables are shared between software and hardware gateways with a pre-downloaded configuration to guarantee the deterministic lookup performance. However, in other systems like TEA [24], cache replacements between software and hardware are involved in the system to fit frequently used entries to the hardware switch. Currently, we do not prefer the cache-based design to avoid cache breakdown and sudden performance degradation in some extreme cases. We follow "Occam's razor" to protect the simplicity and reliability of our system.

Metadata tweaks for complex cloud services. Cloud services are complex and we need to use a large amount of metadata to store the intermediate results during the pipelined table lookups. However, we notice that the on-chip PHV (packet header vector) resources where metadata is stored are also scarce, although they have not been exhausted yet. Therefore, how to optimize the table lookup logic as well as the metadata organization to reduce the consumption of PHV is also an interesting topic worth exploring.

7 RELATED WORK

Some previous work has already looked into the design and implementation of the multi-tenant cloud as well as the cloud gateway [8, 25, 38, 45]. As the pioneering work of the multi-tenant cloud, [25] implements a cloud gateway using software to carry thousands of tunnels, which may be sufficient for small-sized cloud vendors but may not work with major cloud vendors. Protego [38] proposes a cloud-scale multi-tenant gateway with software node clustering and flexible tunnel migrating between these nodes, managed by a central controller. Such a scale-out design is surely competent even for large-scale clouds. However, we show in this work that due to the slowdown of CPU improvement, the CPU core is likely to be overloaded by heavy-hitter flows, which has been spotted from time to time in Alibaba Cloud. [8] exposes the memory shortage problem of the cloud gateway to maintain millions of tunnels and proposes a hybrid solution with a commercial switch and a DPDK-based software forwarding node. However, they solve the memory shortage problem simply by using the x86 node to store the virtual routing table with millions of entries. Thus, the majority of traffic will pour into the x86 node. By contrast, Sailfish absorbs the majority of traffic with the programmable switches by compressing major tables into the programmable switches and leaving only the minority of traffic ($< 0.2‰$) to the software. [45] proposes a versatile software gateway, but the forwarding performance is still insufficient for cloud-scale deployment.

Other work [16, 20, 24, 30, 31, 44] builds other stateful network devices such as load balancers, and some of these devices also leverage programmable switching ASICs. They encounter the similar memory shortage problem to maintain large session states but provide quite different solutions. For example, in Silkroad [30], the connection table is compressed by hashing 5-tuples into shorter digests to fit into the Tofino chip. Since load balancers forward traffic based on an exact match of the 5-tuples, hashing is indeed a good option. However, in cloud gateways, longest prefix match is also needed, thus compression simply by hashing is not enough. TEA [24] proposes a cache-based approach, using the Tofino's on-chip memory as the cache and the external DRAM at the x86 nodes as the backup memory. The on-chip memory and the DRAM are connected via fast RDMA channels. Sailfish prefers pre-allocated table entries to the cache-based design in TEA to avoid cache breakdown and sudden performance degradation in extreme cases to protect the stability of gateways and the reliability of cloud services. Another deficiency of TEA is their need for re-circulation, which results in the same throughput cost as our solution.

8 CONCLUSION AND FUTURE WORK

We propose *Sailfish*, a cloud-scale multi-tenant multi-service gateway based on programmable switching ASICs designed for Alibaba Cloud. We share our gains and pains during the deployment of software gateways and explain why we turned to hardware in 2017. To address the memory shortage problem of maintaining large forwarding tables with the Tofino chip, we propose hardware and software co-design for table sharing, horizontal table splitting among gateway clusters and pipeline-aware table compression for a single node. Compared with the software gateway, Sailfish achieves significant single-node performance speedup at roughly the same unit price. The experiences and lessons are also provided.

Currently, after horizontal table splitting among gateway clusters, each gateway cluster carries a portion of entries for some tenants. However, in a real deployment, we observe that not all the entries of tenants are active most of the time. Taking advantage of this, in the future, we plan to build the "N+1" hierarchical XGW-H clusters with N cache clusters at the front serving only active entries and 1 backup cluster storing entries of all tenants to handle the cache miss traffic. The active entries can be identified through data mining or cache replacements. In this way, we can use fewer XGW-H nodes to achieve a much higher processing capability. For example, if only 25% of the tenants' entries are active, we can build 4 cache clusters (each cluster carries the 25% active entries) and 1 backup cluster (the cluster carries the 100% entries) to provide 4x performance at the cost of only 2x the number of XGW-H nodes.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

The authors would like to thank the shepherd Vincent Liu and the anonymous reviewers for their constructive comments.

REFERENCES

[1] 2017. 12.8 Tb/s StrataXGS Tomahawk 3 Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56980-series. (2017).

[2] 2020. Cisco Silicon One. https://www.cisco.com/c/en/us/solutions/silicon-one.html. (2020).

[3] 2020. ECMP Flow-Based Forwarding. https://www.juniper.net/documentation/en_US/junos/topics/topic-map/security-ecmp-flow-based-forwarding.html. (2020).

[4] 2020. High-Capacity StrataXGS Trident4 Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series. (2020).

[5] 2021. Intel Xeon Processor E7 Family. https://www.intel.com/content/www/us/en/products/processors/xeon/e7-processors.html. (2021).

[6] 2021. Practice and thinking of migrating entire Alibaba services to the cloud (in Chinese). https://developer.aliyun.com/article/765369. (2021).

[7] 2021. 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).

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

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

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

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

[12] K Costello and M Rimol. 2021. Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 23% in 2021. *Gartner. Available online: https://www.gartner.com/en/newsroom/press-releases/2021-04-21-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021* (2021).

[13] Tom Coughlin. 2020. Impact of COVID-19 on the consumer electronics market. *IEEE Consumer Electronics Magazine* 10, 1 (2020), 58–59.

[14] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. 2014. Measuring ipv6 adoption. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 87–98.

[15] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2013. Elastras: An elastic, scalable, and self-managing transactional database for the cloud. *ACM Transactions on Database Systems (TODS)* 38, 1 (2013), 1–45.

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

[17] Chongrong Fang, Haoyu Liu, Mao Miao, Jie Ye, Lei Wang, Wansheng Zhang, Daxiang Kang, Biao Lyv, Peng Cheng, and Jiming Chen. 2020. VTrace: Automatic Diagnostic System for Persistent Packet Loss in Cloud-Scale Overlay Network. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 31–43.

[18] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices* 47, 4 (2012), 37–48.

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

[20] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2014. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 27–38.

[21] Gary Garrison, Sanghyun Kim, and Robin L Wakefield. 2012. Success factors for deploying cloud computing. *Commun. ACM* 55, 9 (2012), 62–68.

[22] Stephen D Goglin and Linden Cornett. 2009. Flexible and extensible receive side scaling. (Sept. 1 2009). US Patent 7,584,286.

[23] Christian Hopps et al. 2000. *Analysis of an equal-cost multi-path algorithm*. Technical Report. RFC 2992, November.

[24] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. 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*. 90–106.

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

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

[27] Mallik Mahalingam, Dinesh G Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. 2014. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. *RFC* 7348 (2014), 1–22.

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

[29] Christopher McCarthy, Kevin Sullivan, and Rejith Krishnan. 2013. Systems and methods for private cloud computing. (July 23 2013). US Patent 8,495,611.

[30] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. 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*. 15–28.

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

[32] Siyi Qiao, Chengchen Hu, Gordon Brebner, Jianhua Zou, and Xiaohong Guan. 2020. Adaptable Switch: A Heterogeneous Switch Architecture for Network-Centric Computing. *IEEE Communications Magazine* 58, 12 (2020), 64–69.

[33] Martin Raab and Angelika Steger. 1998. "Balls into bins"—A simple and tight analysis. In *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 159–170.

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

[35] Theodoros Rokkas, Ioannis Neokosmidis, and Ioannis Tomkos. 2018. Cost and Power Consumption Comparison of 400 Gbps Intra-Datacenter Transceiver Modules. In *2018 20th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 1–4.

[36] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–14.

[37] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*. 164–176.

[38] Jeongseok Son, Yongqiang Xiong, Kun Tan, Paul Wang, Ze Gan, and Sue Moon. 2017. Protego: Cloud-scale multitenant ipsec gateway. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 473–485.

[39] Neil C Thompson and Svenja Spanuth. 2021. The decline of computers as a general purpose technology. *Commun. ACM* 64, 3 (2021), 64–72.

[40] Henry Wang. 2019. Algorithmic longest prefix matching in programmable switch. (Dec. 17 2019). US Patent 10,511,532.

[41] Timothy Wood, Prashant J Shenoy, Alexandre Gerber, Jacobus E van der Merwe, and Kadangode K Ramakrishnan. 2009. The Case for Enterprise-Ready Virtual Private Clouds. In *HotCloud*.

[42] Beibei Wu, Yang Xu, Hongbin Lu, and Bin Liu. 2005. A practical packet reordering mechanism with flow granularity for parallelism exploiting in network processors. In *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 8–pp.

[43] Nofel Yaseen, Behnaz Arzani, Ryan Beckett, Selim Ciraci, and Vincent Liu. 2020. Aragog: Scalable Runtime Verification of Shardable Networked Systems. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 701–718.

[44] Jiao Zhang, Shubo Wen, Jinsheng Zhang, Hua Chai, Tian Pan, Tao Huang, Linquan Zhang, Yunjie Liu, and F Richard Yu. 2020. Fast Switch-Based Load Balancer Considering Application Server States. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1391–1404.

[45] Menghao Zhang, Jun Bi, Kai Gao, Yi Qiao, Guanyu Li, Xiao Kong, Zhaogeng Li, and Hongxin Hu. 2019. Tripod: Towards a scalable, efficient and resilient cloud gateway. *IEEE Journal on Selected Areas in Communications* 37, 3 (2019), 570–585.

[46] Peng Zheng, Arvind Narayanan, and Zhi-Li Zhang. 2019. A closer look at NFV execution models. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*. 85–91.