Achelous: Enabling Programmability, Elasticity, and Reliability in Hyperscale Cloud Networks

Chengkun Wei[†*], Xing Li[†§*], Ye Yang[§†*], Xiaochong Jiang[†], Tianyu Xu[†], Bowen Yang[§], Taotao Wu[§], Chao Xu[§], Yilong Lv[§], Haifeng Gao[§], Zhentao Zhang[§], Zikang Chen[§], Zeke Wang[†], Zihui Zhang[†], Shunmin Zhu[‡§□], Wenzhi Chen[†□]

[†]Zhejiang University [§]Alibaba Cloud [‡]Tsinghua University

ABSTRACT

Cloud computing has witnessed tremendous growth, prompting enterprises to migrate to the cloud for reliable and on-demand computing. Within a single Virtual Private Cloud (VPC), the number of instances (such as VMs, bare metals, and containers) has reached millions, posing challenges related to supporting millions of instances with network location decoupling from the underlying hardware, high elastic performance, and high reliability. However, academic studies have primarily focused on specific issues like high-speed data plane and virtualized routing infrastructure, while existing industrial network technologies fail to adequately address these challenges.

In this paper, we report on the design and experience of *Achelous*, Alibaba Cloud's network virtualization platform. *Achelous* consists of three key designs to enhance hyperscale VPC: (*i*) a novel hierarchical programming architecture based on the collaborative design of both data plane and control plane; (*ii*) elastic performance strategy and distributed ECMP schemes for seamless scale-up and scale-out, respectively; (*iii*) health check scheme and transparent VM live migration mechanisms that ensure stateful flow continuity during the failover. The evaluation results demonstrate that, *Achelous* scales to over 1,500,000 of VMs with elastic network capacity in a single VPC, and reduces 25× programming time, with 99% updating can be completed within 1 second. For failover, it condenses 22.5× downtime during VM live migration, and ensures 99.99% of applications do not experience stall. More importantly, the experience from three years of operation proves the *Achelous*'s serviceability, and versatility independent of any specific hardware platforms.

KEYWORDS

Network virtualization, Programmability, Elasticity, Reliability

1 INTRODUCTION

Cloud computing has grown consistently over the years [6, 35], and the promising new cloud services (*e.g.*, serverless computing [40, 46], AI training [38, 52], and digital office [36]) have reinforced this trend. Large-scale enterprises migrate their businesses to the cloud, seeking flexible scalability in response to changing business demands, which often results in network peak-to-valley phenomena. Simultaneously, tenants expect network services in the cloud to be as reliable as physical hardware. These comprehensive demands pose significant challenges in the design of cloud networks. However, academic studies mainly focus on specific issues such as high-speed data plane [48, 49] and virtualized routing infrastructure [31, 41], while existing industrial network technologies struggle to adequately support hyperscale cloud networks. For example, Andromeda [14] presents Hoverboard to provide high performance, isolation, and velocity at scale in Google cloud network virtualization. Nevertheless, these technologies are insufficient in addressing the more significant challenges of network scalability and bursty performance posed by modern cloud applications. Our experiences have taught us the importance of integrating a cohesive system and distributing functionality across network infrastructures, such as controllers, gateways, and vSwitches. In the following, we highlight three primary challenges encountered in modern cloud environments.

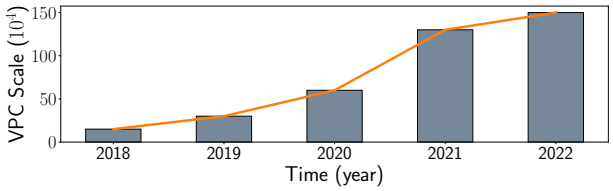


Figure 1: Alibaba e-commerce VPC scale expansion over the years.

Challenge 1: Sub-second reconfiguration time when serving millions of concurrent instances. The first significant challenge stems from the migration of e-commerce businesses (*e.g.*, Taobao [5]) to the cloud, leading to the deployment of a vast number of instances (e.g., VMs, bare metals, and containers) within a single Virtual Private Cloud (VPC). Figure 1 illustrates a prime example of the exponential growth of Alibaba's e-commerce instances within a single VPC, reaching 1,500,000 instances in 2022. The hyperscale network exhibits two key characteristics: *high deployment density* and *frequent creation/destruction of instances*. For example, during traffic peaks, we may need to initiate an additional 20,000 container instances, each having a lifecycle of only a few minutes.

The existing network programming methods are not equipped to handle the creation and readiness of such an enormous number of instances concurrently. For example, Andromeda [14] presents a design to scale 100,000 instances in the paper, and AWS supports up to 256,000 instances per VPC according to the latest report [3]. However, within Alibaba Cloud, the number of instances that need to be supported is an order of magnitude higher than any existing technology can accommodate. The proliferation of these hyperscale VPCs leads to a substantial increase in routing entries and a high frequency of network changes, thereby challenging the network convergence rate. Specifically, the cloud vendors must provide

*Co-first authors □Co-corresponding authors

services within a limited timeframe, such as the ability to launch numerous serverless containers with ready network connectivity within 1 second.

Challenge 2: High elastic network for heavy-traffic middlebox deployment. The second significant challenge arises from migrating traditional heavy-load network applications, such as middleboxes, to virtual machines (VMs) in the cloud. Initially, these middleboxes (*e.g.*, load balancer, NAT gateway, and firewall) were deployed on dedicated hardware within the physical network. Migrating them to the cloud offers elasticity and enterprise cost optimization advantages. However, this transition disrupts the isolation provided by the originally dedicated devices, potentially leading to resource contention (both network and CPU resources) with other instances co-located on the same host. Furthermore, the middlebox VMs must exhibit elastic scalability (both scaling up and scaling out) to effectively handle variable traffic pressures, as their customers' capacity demands are subject to change.

Unfortunately, existing studies [8, 11, 22, 27, 45] fail to account for resource competition and are thus unsuitable for providing isolated yet elastic resource allocation on multi-tenant cloud hosts. In terms of scale-out scenarios, load-balance approaches [15, 55] and traditional Equal-Cost Multi-Path (ECMP) routing mechanisms [9, 37] introduce new bottlenecks. The centralized load balancers and ECMP forwarding nodes emerge as primary limiting factors in network scalability, as they handle traffic originating from millions of source VMs.

Challenge 3: High cloud serviceability and reliability. For cloud vendors, resource utilization and reliability serve as crucial indicators of a cloud platform. However, in the cloud network, detecting failures and achieving fast failover present significant challenges. The dynamic mapping between the virtual network and physical devices makes it difficult to establish a precise topology, which in turn hampers failure telemetry. Existing failure telemetry technologies [8, 11, 22, 27, 32, 34, 45] either focus on physical networks or lack real-time capabilities, thus failing to guarantee reliability in the virtual network. Compounding the issue, most live migration technologies [28, 30, 53] disregard the traffic continuity of stateful flows, resulting in disruptions to tenant services.

With the above challenges in mind, we set out to re-architect our network architecture three years ago. We identify that current network components (*i.e.*, controller, gateway, and vSwitch) cannot independently meet these requirements. Leveraging insights from both the control and data planes, we propose *Achelous*, Alibaba Cloud's network virtualization platform that offers programmability, elasticity, and reliability in hyperscale cloud networks. We focus on introducing three main design choices that build on our extensive operational experience.

First, to solve the challenge of the larger forwarding table and slower convergence rate, we propose a novel programming mechanism, which is actively learning the forwarding information on demand from the gateway instead of the controller. The controller only needs to offload network rules to the gateway instead of the vSwitch on each host (§4.1). We also design a lightweight forwarding cache with effective management on IP granularity, so as to further narrow down the table structure (§4.3). **Second**, to achieve scalability within the host while ensuring performance isolation, we propose a novel elastic strategy that balances the isolation and the burst traffic, and achieves high utilization of bandwidth and CPU resources (§5.1). Moreover, we demonstrate a distributed ECMP mechanism redirecting traffic to multiple vSwitches to enable seamless scale-out of services among hosts (§5.2). **Third**, we present a link health check scheme to validate the network link status among vSwitch and VMs, along with a monitor to detect the status of vSwitch itself (§6.1). Furthermore, we introduce a series of transparent VM live migration techniques, including traffic redirect, session reset, and session sync. These techniques condense VM migration downtime, supporting the continuity of both stateless and stateful flows with applications' unawareness (§6.2).

Our deployment and evaluation results validate the efficacy of these design choices. *Achelous* has acted as the cornerstone of Alibaba's VPC network stack for years. It significantly improves the user-perceivable experience. Notably, 99% of services exhibit a startup delay of less than 1 second, while 99.99% of applications experience no stalls. After years of operation, we believe that the design choices of *Achelous* are not only feasible but also highly effective, and the lessons learned (§8) can be broadly applied to other cloud vendors.

Thus this paper makes the following contributions:

- We propose a novel on-demand active learning programming mechanism with an optimized table structure to speed up the network coverage for hyperscale VPCs. The VPC with more than 1.5 million VM instances can complete the configuration coverage within 1.33*s*. Compared with traditional deployment patterns, our mechanism improves the configuration convergence time by more than 25x.
- We present an elastic network capacity strategy and distributed ECMP mechanism, which supports scale-up on the premise of isolation and seamless scale-out among hosts. For the heavy-loaded services, we support seamless expansion and contraction within 0.3*s*.
- We design a whole failure detection and avoidance mechanism for VMs, which includes abundant health check methods and seamless live-migration schemes. With these mechanisms, VM's failover latency is in the order of 100*ms*, without impacting applications inside the guest VM.

2 BACKGROUND AND MOTIVATION

Achelous is Alibaba Cloud's network virtualization platform, which has evolved over the past decade to support Alibaba Cloud's virtual networks. It has undergone significant improvements in performance through the phases of *Achelous* 1.0 and *Achelous* 2.0. As cloud networks continue to scale, new challenges have arisen, leading us to enhance and iterate *Achelous* 2.0. In this section, we discuss the evolution of *Achelous*, the VM network datapath in *Achelous* 2.0, and the new challenges.

2.1 The role of *Achelous* in Alibaba Cloud

In Alibaba Cloud, *Achelous* provides VM network virtualization via three fundamental components (shown in Figure 2): the SDN controller on the control plane, the vSwitch and gateway on the data plane.

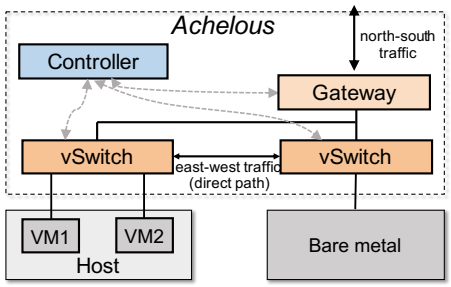


Figure 2: *Achelous* in Alibaba Cloud

On the control plane, the controller manages all the network configurations during the instance (*e.g.*, VM, bare metal) life cycles, and issues network rules into vSwitch and gateway. For example, once a VM instance (say VM1) is created, the controller will issue all existing VM1's network forwarding information into the vSwitch and gateway. Then, if the VM1's network changes (such as migrating to another host, a new network card is mounted), the controller will update and correct the corresponding rules in the data plane.

On the data plane, the vSwitch serves as a per-host switching node dedicated to VM traffic forwarding. The gateway, acting as a higher-level forwarding component, facilitates interconnection between different domains. In Figure 2, we provide an example: when VM1's packet is processed by the vSwitch, it determines the appropriate destination. If the destination VM is on the same host as VM1 (e.g., VM2), the vSwitch directly forwards the packets. Otherwise, if the destination VM is on a different host (e.g., bare metal), the packets are relayed through the gateway. For more details about the gateway, please refer to Sailfish [42], which supports deployment on various hardware platforms.

2.2 The Evolution of *Achelous*

The development of *Achelous* began over ten years ago with its initial version, *Achelous* 1.0, providing the foundational virtual network environment for Alibaba Cloud. Since it was developed earlier than the formulation of the VXLAN standard [39], *Achelous* 1.0 experienced the architecture transformation from the classic layer-2 network to the standard VPC overlay network. This evolution enhanced security by enabling layer-2 isolation of guest VMs through VXLAN Network Identifier (VNI). However, *Achelous* 1.0 faced performance challenges due to its data plane relying on the netfilter [4] in the Linux kernel, which is always the bottleneck of network processing.

In *Achelous* 2.0, data plane performance has been significantly improved through Data Plane Development Kit (DPDK) [2] acceleration and hardware offloading. These acceleration methods effectively reduce the packet copying overhead on the datapath, which is critical to the forwarding performance. Additionally, the optimization in *Achelous* 2.0 involves offloading east-west traffic (VM-VM traffic) to alleviate potential bottlenecks. Since the east-west traffic constitutes over 3/4 of the total traffic, relying on the gateway for relaying can introduce noticeable bottlenecks. In *Achelous* 2.0, the controller issues all the east-west rules to the vSwitches, so that the vSwitch can forward east-west traffic via *direct path* (see Figure 2). Nevertheless, it will cause the *challenge1* (in §2.4) because each vSwitch should be notified of the correct rules when the network changes.

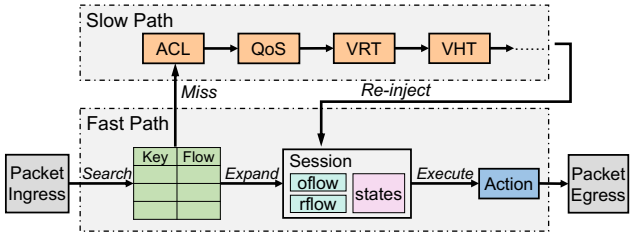


Figure 3: VM network datapath in *Achelous* 2.0

2.3 The VM network datapath in *Achelous* 2.0

To provide insight into the platform, we briefly introduce the key procedures and data structures on the VM network datapath of *Achelous* 2.0.

The VM network datapath of Achelous 2.0 (Figure 3) consists of two parts: the slow path and the fast path. The slow path encompasses a packet processing pipeline consisting of various tables, each serving a specific function. These tables include the Access Control List (ACL), Quality of Service table (QoS), the VXLAN Routing Table (VRT), VM-HOST mapping table (VHT) (*i.e.*, vm_ip-host_ip mapping relationship) and so on. All tables are configured by the controller, with VHT being particularly crucial. As the number of VMs escalates within the VPC, the VHT encounters significant expansion, resulting in a surge of entries.

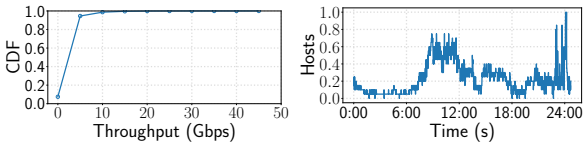
As for the fast path, we first introduce **a new data structure called *session***, consisting of a pair of flow entries in two directions (*i.e.*, *oflow* for the original direction and *rflow* for the reverse direction) and all the states needed for packet processing. The flow entry contains five-tuple of a packet and adopts the exact matching algorithm. The packet processing procedures are as follows: 1) the first packet is processed through the pipeline of the slow path; 2) then a five-tuple flow entry and its session will be generated and re-injected into the fast path; 3) subsequent packets can be directly matched and processed on the fast path.

The performance gap between the fast path and slow path in *Achelous* 2.0 is significant, with the fast path exhibiting a performance advantage of 7-8 times over the slow path. This results in varied network-dedicated CPU consumption and performance for packets from different flows. For example, VMs with short-lived connections may monopolize up to 90% of vSwitch CPU resources, impacting other VMs. This intensifies the *challenge2* (in §2.4).

Moreover, the demand for hyperscale VPC networks amplifies the challenges posed by perceived design "flaws" in the data plane and control plane of *Achelous* 2.0, particularly in adapting to new service scenarios like e-commerce and middlebox migration to the cloud.

2.4 Challenges to *Achelous* 2.0

Compared with the limited VPC network scale managed by cloud vendors, the new scenarios under hyperscale deployment bring three severe challenges:



(a) Average VM throughput distribution **(b) Hosts with high CPU competition**

Figure 4: Unpredictable network capacity demands

(1) Larger Routing Tables and Slower Convergence Rate. One of the most critical considerations is *control plane scalability*. On the one hand, the expansive nature of large networks necessitates larger routing tables, such as VHT and VRT, resulting in increased memory consumption on resource-constrained hosts or Cloud Infrastructure Processing Units (CIPUs) [1]. For example, in Alibaba Cloud, VPCs can accommodate over 1.5 million VMs, thereby leading to a substantial number of table entries and consuming multiple gigabytes of memory for efficient packet forwarding. On the other hand, there is the matter of managing the high volume of concurrent entry change requests. Our empirical analysis reveals that the control plane receives more than 100 million network change requests per day. The significant influx of requests challenges network convergence, a crucial factor for autoscaling and failover.

(2) Balancing Idle Resources and Burst Traffics on Fairness and Isolation. With the increasing deployment density of VMs, there is an urgent need for *elastic network capabilities*. An analysis conducted in Alibaba Cloud reveals the following findings: 1) the average throughput of over 98% of VMs is below 10 Gbps, indicating significant network resource idleness (see Figure 4a); 2) however, *network bursting* occurs daily, leading to competition for bandwidth and CPU resources in the data plane (see Figure 4b[1]). For instance, online meeting services experience traffic bursts during work hours while requiring minimal bandwidth during breaks. To balance idle resources and burst traffic, achieving high elasticity becomes imperative in resource allocation while upholding fairness and isolation. Moreover, traffic-heavy tenants demand seamless scaling out capabilities across multiple hosts when facing flow floods.

(3) Detecting Network Risks and Escaping with Tenant Unawareness. With the increasing complexity of VPC network configurations, ensuring high *network reliability* is crucial. Traditionally, network operations and maintenance heavily rely on human intervention, resulting in a lack of predictive failure capabilities. Network failures tenants report often entail significant time and effort to locate and resolve. Although we have developed various network failure localization technologies [16], advanced preemptive failure detection remains a challenge. In the context of hyperscale VPCs, adopting network risk awareness approaches becomes increasingly crucial for early failure detection and avoidance, enabling uninterrupted service delivery to tenants. Furthermore, transparent live migration technology is critical in ensuring traffic continuity when assisting tenant VMs in transitioning from faulty hosts or networks.

The above challenges motivate us to rethink the design of data plane and control plane in *Achelous*, and make new innovations.

[1]We counted the hosts that data plane CPU usage exceeding 90% in typical regions during one day. The figure has been normalized.

3 DESIGN OVERVIEW

To tackle the challenges posed by scale explosion, we propose innovative designs and advancements in *Achelous* 2.1. Departing from the principle of over-optimization for individual components, we explore synergistic collaborations between the data plane and the control plane. The three key improvements are as follows:

Hyperscale Network Programmability. In order to reduce the network convergence time and improve the memory efficiency, we propose an *active learning mechanism* (§4.1) in *Achelous* 2.1. In this mechanism, the vSwitch only manages a forwarding cache and actively learns route information from gateways via a Route Synchronization Protocol (RSP). So the controller only needs to program network for the gateway, rather than each vSwitch node. This approach enables rapid network updates through the high-performance data plane and ensures synchronization to affected vSwitches with minimal convergence time. As a result, we avoid storing complete forwarding information on every host, which improves per-server memory utilization and scalability by over an order of magnitude.

Elastic Network Capacity. *Achelous* conducts seamless scale-up and scale-out methods to achieve high elastic network performance based on isolation. In the data plane, network capacity is provided through the dedicated CPU cores, necessitating the consideration of all available resources. *Achelous* 2.1 adopts *elastics strategy* (§5.1) to allocate both bandwidth and CPU resource for all the VMs within a host, which not only makes full use of idle resources for bursty traffic but also guarantees performance isolation. Additionally, to provide seamless service expansion among hosts, we implement a distributed ECMP mechanism (§5.2) in each vSwitch (the edge node of *Achelous*), eliminating forwarding bottlenecks associated with centralized deployment in the underlay network.

Network Risk Awareness and Live Migration. *Achelous* performs network link health checks to active perception and early warning faults such as network congestion or VM halt (§6.1). This involves vSwitches sending periodic health check packets to VMs based on a pre-configured checklist. Simultaneously, the vSwitch itself reports performance statistics to the controller, enabling active intervention to mitigate network risks. Once the risks are detected, the vSwitch provides transparent VM live migration for failover under the controller's guidance. During migration, *Achelous* implements the traffic redirect, the session reset, and the session copy technologies (§6.2) to fulfill the network properties (such as low downtime, stateless flows, stateful flows, and application unawareness) for uninterrupted service.

4 HYPERSCALE NETWORK PROGRAMMING

In this section, we present the designs of *Achelous* 2.1 that target the first challenge in §2.4. We detail our programming mechanism developed for hyperscale cloud networks, named Active Learning Mechanism (ALM).

4.1 Design of Active Learning Mechanism

Problem & Goal. High memory consumption remains a critical challenge for large-scale forwarding tables. In software forwarding architectures [17, 43], vSwitches share the memory with VMs,

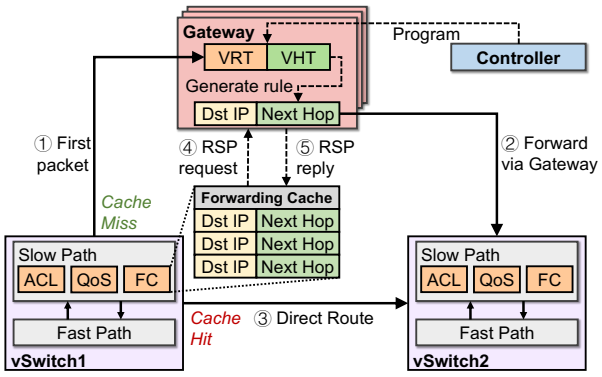


Figure 5: Active Learning Mechanism

resulting in significant memory usage for in-memory hyperscale forwarding entries, potentially straining host memory resources. In hardware-accelerated architectures [18, 44], the limitations of available high-speed on-chip memory on dedicated hardware further exacerbate the memory resource constraints. Even worse, when the large-scale forwarding tables encounter high concurrent network change requests, the controller cannot notify each affected vSwitch in time and thus will become a bottleneck. Thus, we aim to develop an effective programming mechanism in *Achelous* to serve the hyperscale network.

Insight. Our key observation is that data center networks undergo constant changes to handle failures, deploy/upgrade services, and adapt to traffic fluctuations. Consequently, the locations of VMs or containers may change frequently due to VM migration, failures, and creation/release events. Specifically, the Virtual Routing Table (VRT) and VM-Host mapping table (VHT) experience high-frequency updates. On the other hand, configuration tables on the vSwitch, such as ACL and QoS, change less frequently. For example, tenant security group configurations remain relatively stable, while the VHT undergoes updates during service expansion and contraction. Thus, we can shift the frequently changed table to the powerful gateway so as to reasonably reduce the resource consumption of vSwitch and improve the overall efficiency.

Design Overview. We continue to iterate on our *Achelous* design based on long-term production experience and observation as a cloud vendor. The ALM is our latest move towards hyperscale VPC for supporting millions of VMs. As shown in Figure 5, the ALM entails modification on all three core components in *Achelous*. The core idea of ALM is to free the controller from the heavy load of issuing network changes, and let vSwitch actively synchronize forwarding rules via the gateway on demand. Accordingly, we design: 1) a light-weight forwarding table, named as *Forwarding Cache* (FC) table for higher storage efficiency; 2) an on-demand forwarding rules synchronization mechanism for faster convergence.

4.2 Light Weighted Forwarding Table with Hierarchy Packet Processing Paths

Light Weighted Forwarding Table. We introduce a lightweight Forwarding Cache (FC) table for efficient software switching. Instead of relying on explicit VRT/VHT entries, the vSwitch utilizes the compact "Dst IP -> Next Hop" mappings learned from gateways. First, the IP-based entries are more compact and can cover larger traffic because multiple flows of each VM-VM pair may reuse only one entry. We can reduce the flow table entries of different port numbers to a single IP entry, which may be 65535 times less storage in extreme cases. Second, by shifting from a flow-based table to an IP-based table, we address the vulnerability to Tuple Space Explosion (TSE) attack, which is a denial-of-service attack against the software packet classifier [13].

Hierarchy Packet Processing Paths. As shown in Figure 5, upon parsing the packet's Dst IP of the VXLAN inner header, the subsequent packet processing follows the hierarchy paths: 1) *Fast path*: the fast path keeps the same as in §2.3, and acts as a service-logic-irrelevant acceleration path for high performance. The unmatched packets in the fast path will be upcalled to the slow path; 2) *Slow path*: the original VHT and VRT on the slow path (see §2.3) are replaced by FC, but the ACL and QoS tables are still preserved. For management, the slow path updates FC table entries on demand from the gateway to get a local small table, which only requires a small storage space. Specifically, the slow path looks up the FC according to the "Dst IP" of each packet. If cache misses, packets will be upcalled to the gateway ①, and eventually forwarded to the destination ②. While the packets that hit in the FC will be re-injected to the fast path and routed directly ③.

The hierarchical packet processing design in *Achelous* simplifies the forwarding table structure and the packet processing logic within the vSwitch. By leveraging the gateway to store and manage the complete set of forwarding rules, the vSwitch can synchronize entries on demand, significantly reducing the storage overhead for each vSwitch node. This design approach also decouples the vSwitch from intricate routing logic, resulting in improved forwarding performance, enhanced versatility, and simplified maintenance.

4.3 On-Demand Forwarding Rules Synchronization Mechanism

Active Traffic-driven Learning From the Gateway. In addition to its role in the data plane, the gateway in our architecture also functions as a forwarding rule dispatcher in the control plane. The vSwitch nodes learn the forwarding rules on demand from the gateway via the Route Synchronization Protocol (RSP), our in-house-designed protocol. As shown in Figure 6, RSP has two types of packets: request and reply. The request packet contains flow's five-tuple, and the reply packet contains the next hops for the corresponding request. vSwitch determines whether to learn rules or directly send traffic to gateway based on factors such as flow duration, throughput, etc. When the vSwitch needs to learn a rule, it constructs an RSP request packet and sends it to the gateway. Then the gateway parses the request, collects specific rules, and writes to the reply packet. Upon receiving the reply, the vSwitch inserts the corresponding entries into the Forwarding Cache (FC).

Synchronization Frequency. To ensure the timeliness of existing entries in vSwitch's FC that learned from gateway, ALM adopts a periodic update strategy. We created a management thread in the vSwitch to traverse FC entries every 50*ms*, for checking whether the lifetime (the interval between the last update and the present) of

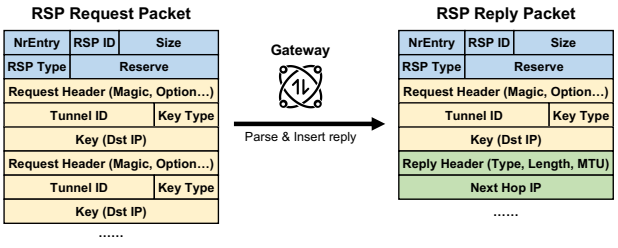


Figure 6: Format of Route Synchronization Protocol

each entry exceeds a certain threshold (*e.g.*, 100*ms*). If the lifetime exceeds the threshold, the vSwitch proactively sends RSP requests ④ for data reconciliation with the gateway. Then, based on the RSP reply from gateway⑤, vSwitch performs the appropriate actions on the local FC. For example, if an entry on the gateway is changed or deleted, the vSwitch will update the corresponding local FC entry. If the data reconciliation indicates that the local FC is up-to-date, the vSwitch will not operate on FC.

Reducing Overhead. To reduce the number of RSP packets in the network and improve the efficiency of ALM, we adopt batch processing design in ALM. In vSwitch, we allow multiple query requests to be encapsulated into a single RSP packet. Correspondingly, the gateway can also implement batch processing and encapsulate multiple responses into one reply packet. We verify our design in the deployment and get the results that the average request packet length is about 200 bytes and the maximum bandwidth share of RSP < 4% (see §7.1). So this overhead is acceptable compared to the more powerful functions it brings for the virtual networks (*e.g.*, we can negotiate the MTU, encryption capabilities, and other features for tenant's connections when necessary via RSP protocol).

5 ELASTIC NETWORK CAPACITY

In this section, in order to solve the second challenge in *Achelous* 2.0 (see §2.4), we first introduce the scale-up scheme within a single host, which provides elasticity on the premise of performance isolation. Next, we present our scale-out scheme within a cluster of hosts.

5.1 Scale-up within Host

Problem & Goal. The elastic bandwidth was extensively studied [22, 27, 32, 34], however, only monitoring bandwidth can not meet the elasticity requirements. For example, when a VM comes to a burst of short connections, although the VM's bandwidth does not reach the superior limit, it may consume too many CPU resources of the vSwitch. Other VMs on the same host can not get enough CPU resources, which leads to bandwidth decline and a breach of isolation within the host. Thus, we need to provide elasticity on the basis of isolation to VMs within a host.

Elasticity and Isolation. To solve this issue, we propose a credit strategy monitoring two dimensions of indicators. The *first* indicator is BPS/PPS[2] of the VM, denoted by the R^B, which directly limits the VM's traffic rate. The *second* indicator is the CPU cycle used by the vSwitch to transfer traffic for the VM, denoted by the R^C. Both indicators provide systematic assurance of elasticity and isolation for each VM.

[2]Abbreviation for Bits-Per-Second and Packets-Per-Second correspondingly.

Algorithm 1 Elastic Credit Algorithm

Input: $R_{base}, R_{max}, R_\tau, R_T, Credit_{max}, \lambda, m$

$Credit = 0$
while $True$ **do**
  if $R_{vm} \le R_{base}$ and $Credit < Credit_{max}$ **then**
    $Credit = Credit + (R_{base} - R_{vm})$ ▷ Accumulating
    if $Credit > Credit_{max}$ **then**
      $Credit \leftarrow Credit_{max}$
    end if
  else $R_{vm} > R_{base}$
    if $R_{vm} > R_{max}$ **then**
      $R_{vm} \leftarrow R_{max}$
    end if
    Calculate $\sum_{\forall} R_{vm}$ and Get Top-k set $\mathbb{T}_k$
    if $\sum_{\forall} R_{vm} > \lambda \cdot R_T$ and current VM $\in \mathbb{T}_k$ **then**
      $R_{vm} \leftarrow R_\tau$
    end if
    $Credit = Credit - (R_{vm} - R_{base}) \times C$ ▷ Consuming
  end if
  Sleep(m)
end while

Elastic Strategy. We present an elastic credit algorithm to achieve high utilization of both bandwidth and CPU resources, as shown in Algorithm 1. The key idea is to balance the idle resource and the burst traffic of VMs within a host, and thus the VM can use the idle resources of the host to handle short-term burst traffic.

The total bandwidth resources (CPU resources) for all VMs on the host are denoted by R^B_T (R^C_T). The actual bandwidth resources (CPU resources) usage of the VM is denoted by R^B_{vm} (R^C_{vm}). We set the default resource limit R^B_{base} and R^C_{base} for each VM. In addition, each VM has its own credit values $Credit^B$ and $Credit^C$. The VM can consume or accumulate its credit values. For example, if $R^B_{vm} < R^B_{base}$, in idle state, the vSwitch will accumulate the $Credit^B = Credit^B + (R^B_{base} - R^B_{vm})$ for the VM. If $R^B_{vm} > R^B_{base}$, in burst state, the vSwitch will consume the $Credit^B = Credit^B - (R^B_{vm} - R^B_{base}) \times C$, where $0 < C \le 1$ is the credit consuming rate, for the VM. We defer the details of Algorithm 1 to Appendix A.

Comparison with Token Bucket Method. Our credit algorithm outperforms the token bucket method with stolen functionality. First, there is a specific upper bound on credit consumption, which is one of the important differences between the credit algorithm and the token bucket. Second, the communication overhead of the credit algorithm is lower than the token bucket method, since it does not require the exchange of information between credit buckets. Furthermore, our method can defend against the breach of isolation caused by a large amount of resource occupation for a long time, such as a DDoS attack.

5.2 Scale-out Among Hosts

Problem. Scale-up within a single host cannot keep up with the growing demand of the tenants' service expansion. Therefore, using ECMP to scale-out the service capacity among hosts is imperative.

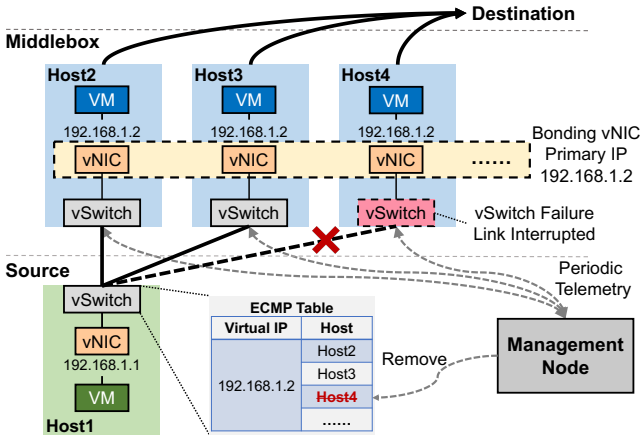


Figure 7: Distributed ECMP. Every vSwitch can realize the ECMP routing without a centralized gateway.

However, the centralized gateway in ECMP deployment will introduce new bottlenecks that hinder network scaling. To this end, we design a distributed ECMP mechanism on every vSwitch node to provide a seamless scale-out.

Distributed ECMP. In this mechanism, the VM is allowed to create a series of *bonding vNICs* for secure inter-VPC communication. As shown in Figure 7, the tenant VM on Host1 creates three *bonding vNICs* and mounts them into the VMs of service VPC (see the VMs on Host2, 3, and 4 of the "Middlebox" VPC). All *bonding vNICs* share a single Primary IP address ("192.168.1.2" in the figure) and the same security group. Then the controller will issue the corresponding ECMP routing entries into the vSwitch on Host1. That ensures tenant VM's packets can be sent to corresponding VMs in "Middlebox" VPC for processing. To scale out smoothly, in the event that the VM resources in the "Middlebox" VPC become exhausted, additional VMs are automatically created and mounted with *bonding vNICs*. Subsequently, the source vSwitch is updated with additional ECMP entries, enabling the redirection of traffic to the newly added VMs. It is important to note that each VM has the ability to be mounted with multiple *bonding vNICs* from different VPCs. This allows the VMs in the "Middlebox" VPC to serve a significantly larger number of VMs from different VPCs.

The distributed ECMP mechanism eliminates potential bottlenecks by ensuring that each source VM is associated with a vSwitch, which effectively redistributes traffic. This approach significantly enhances the elastic capabilities of services with heavy-traffic demands. As an example, Alibaba Cloud's cloud firewall can provide security detection services to millions of tenants by exposing bonding vNIC interfaces on its single VPC. With the horizontal scalability provided by the distributed ECMP mechanism, these tenants can seamlessly benefit from the increased resources without the need to actively manage or adjust to changes in middlebox availability.

Benefits than Load Balance. Even though the LB architectures can provide similar functions as the distributed ECMP mechanism, they often require more tenant configurations. For example, as traffic increases, the centralized LB node may become the bottleneck and needs to scale out, which is accompanied by the change in proxy configuration on the tenant side. In addition, the LB architecture does not support individual security groups for different

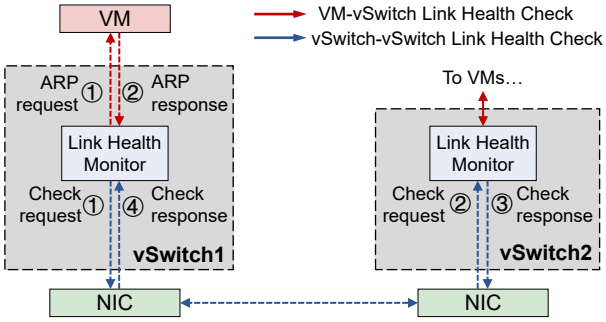


Figure 8: Link health check overview.

tenants. Assuming multiple tenants use the same LB node, but each tenant needs a specific security group, manually configuring these network requirements on the vSwitches behind the LB node is the only option. In contrast, the distributed ECMP mechanism, with its unified configuration of *bonding vNICs*, enables seamless network configuration synchronization and thus makes it easier to scale out for enterprise services.

Failover in Distributed ECMP. To prevent large telemetry traffic of tenant VPCs from blowing up the VMs in service VPC, we leverage a centralized management node for health checks in the distributed ECMP. As shown in Figure 7, the management node periodically telemetries the vSwitches where "Middlebox" VMs locates. Then the management node maintains a global state and synchronizes it with the source side vSwitch. As soon as the vSwitch fails (*e.g.*, vSwitch on Host4), the management node will inform the vSwitch on the source side to update the corresponding ECMP table (*i.e.*, delete the Host4 entry in the ECMP table) to avoid packet loss.

6 NETWORK RISK AWARENESS AND LIVE MIGRATION

In this section, we solve the last challenge of *Achelous* dataplane (see §2.4). We first introduce the network risk awareness scheme. Specifically, we probe the network link connectivity within a host and between hosts, and alert the control plane to the upcoming network risks. Then, we present the seamless live migration schemes for failure recovery.

6.1 Network Risk Awareness

Problem & Goal. Many virtual network stack bugs could not be discovered by previous physical telemetry methods, since physical network probes do not involve virtual network stacks. However, abnormal network events in hyper-scale clouds are frequent and inevitable. Failure to address them in a timely and appropriate manner can lead to minor faults escalating into severe network congestion or application failures. To this end, we design a link health check module on *Achelous* to monitor the status of the hyperscale network for active perception and early warnings of the failures. This module focuses on two types of network risks: 1) network link health which consists of VM-vSwitch, vSwitch-vSwitch, and vSwitch-gateway links; and 2) virtual network device status information, which indicates the running status of the network device itself. We introduce the design details of both as follows.

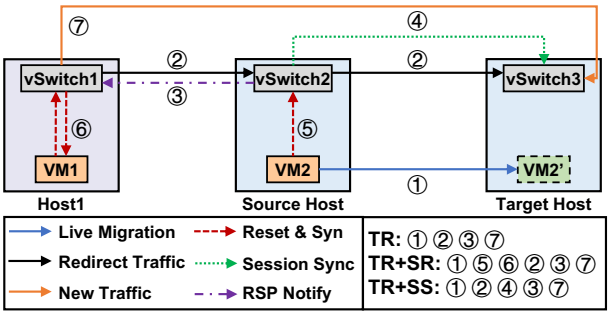


Figure 9: Live Migration. TR = Traffic Redirect, SR= Session Reset, and SS = Session Sync.

Link Health Check. As shown in Figure 8, the VM-vSwitch health check and the vSwitch-vSwtich health check are the red path and the blue path, respectively. The VM-vSwitch denotes the link from vSwitch to VMs in the host. The vSwitch sends ARP requests to the VMs and gets the VMs' ARP response to check the VMs' network status. The vSwitch-vSwitch is the link from one vSwitch to the vSwitch on other hosts. After the monitor controller system configures a checklist (*i.e.*, IP address), the link health check module sends health check packets to the VMs in the checklist. Then link health monitor analyses the responses' latency and reports risks (*e.g.*, VM failure and link congestion) to the control plane. To minimize the intrusion of health check packets into the data plane, we set the health check frequency to 30*s* to reduce additional overheads. Meanwhile, the *Achelous* encapsulates health check packets in a specific format and forwards them only to the link health monitor.

Device Status Health Check. In addition to checking the health status of links, the *Achelous* also checks virtual network devices' health status. First, the *Achelous* monitors device's CPU load and memory usage. Meanwhile, the *Achelous* monitors the network performance, such as the packet loss rates of virtual and physical NICs. If a network device is risky (*e.g.*, high CPU load, high NIC drop rate, and memory exhaustion), we will report these anomalies to the controller. Then, the controller will intervene and start the failure recovery mechanism. The health check mechanism allows the network to be risk-aware, altering potential risks, and proactively intervening the risk.

6.2 Transparent VM Live Migration

Problem & Goal. We perform failure recovery through live migration. Traditional live migration mechanisms[12], however, do not consider the stateful traffic continuity and tenant unawareness. To solve this challenge, we first introduce four properties to guarantee traffic continuity during VM live migration. Then, we show the live migration schemes evolution in *Achelous* to meet these properties step by step.

Properties for VM Live Migration. As shown in Table 1, the live mitigation for failure recovery should meet the following properties: 1) *Low downtime* means that live migration should achieve continuous high throughput and millisecond-level downtime. Second-level downtime cannot meet the needs of hyperscale scenarios; 2) *Stateless flows* refer that we should fast redirect stateless flows (*e.g.*, UDP and ICMP); 3) *Stateful flows* mean that the migration scheme supports the stateful flows (*e.g.*, TCP and NAT) and adaptive processing of the flow state, session information, and even the states of ACL security group; 4) *Application unawareness* means that the migration schemes should adapt to various applications, and native applications do not need to be aware of the migration process.

Table 1: The properties of live migration schemes.

Method	Low downtime	Stateless flows	Stateful flows	Application unawareness
No TR	×	✓	×	×
TR	✓	✓	×	×
TR+SR	✓	✓	✓	×
TR+SS	✓	✓	✓	✓

The VM Live Migration Schemes. We first design a *Traffic Redirect* (TR) scheme to forward the stateless flows and meet the low downtime requirement. To extend the TR scheme to ensure the continuity of stateless flows, we propose *Session Reset* (SR) scheme. However, the SR scheme needs to modify the client application in VM to respond to TCP reconnection requests during the live migration, which reduces vendors' service compatibility and quality. Last, we develop *Session Sync* (SS) in *Achelous*, which synchronizes necessary *sessions* on-demand and keeps connection states during migration progress. The SS scheme ensures stateful flow continuity with native applications' unawareness. We present all these schemes in Figure 9. Due to space limitation, we defer the live migration steps of these schemes in detail to Appendix B.

Our live migration scheme only has milliseconds of downtime. So that *Achelous* dataplane can fast recover both stateless and stateful flows without awareness of native client services. Based on health monitoring and failure warning, we can smoothly migrate VMs to other hosts to avoid possible failures or quickly recover from failure, which greatly improves the reliability of the cloud network.

7 EVALUATION

In this section, we first present the ALM performance in the real-world deployment. Then, we evaluate the effectiveness and robustness of the elastic credit algorithm. Last, we measure the downtime and flow continuity of the live migration schemes. In our evaluation, we collect the data in five typical regions of Alibaba Cloud where it has deployed *Achelous*. These regions' scale range from hundreds to tens of millions of instances.

7.1 Effectiveness of ALM

Programming Time. Figure 10 shows the programming time of *Achelous* in different scale regions. We can see that: 1) the ALM has low convergence time in our production scenarios. For example, the average programming time is 1.334s under in VPC with 10^6 VMs, while the baseline programmed-gateway model is 28.5s, which is 21.36× larger than ALM; 2) the ALM has better scalability. With the number of VMs rising from 10 to 10^6, the preprogrammed-gateway models' average programming time changes from 2.61s to 28.50s, which increases 10.9 times. However, the ALMs' average programming time increases from 1.03s to 1.33s, which only introduces 0.3s.

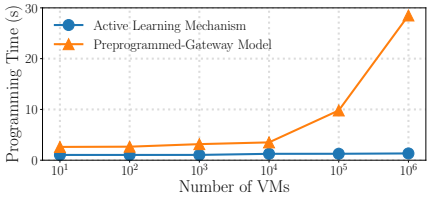


Figure 10: The Programming time of ALM in different cloud regions.

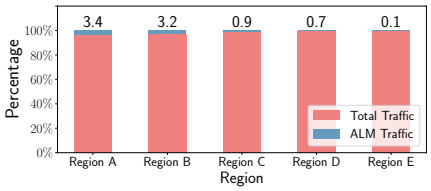


Figure 11: The ratio of ALM traffic in different cloud regions.

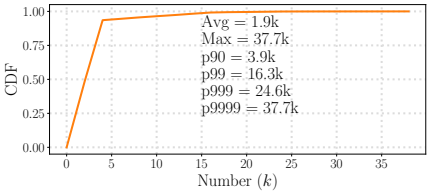


Figure 12: Cumulative distribution of the FC table entries in VPC.

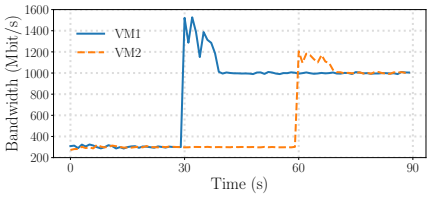


Figure 13: Bandwidth performance of elastic credit algorithm.

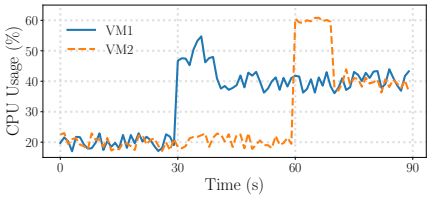


Figure 14: CPU usage performance of elastic credit algorithm.

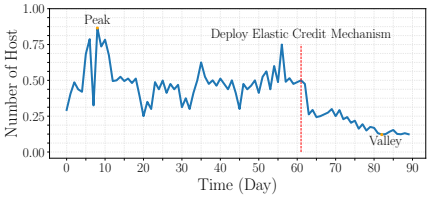


Figure 15: The hosts suffering resources contention (normalized).

That shows *Achelous* has the ability to support network scale with a higher order of magnitude.

ALM traffic and FC entries. Figure 11 shows the proportion of ALM traffic of different regions whose network scales range from hundreds of Gbps to tens of Tbps. We can see that: 1) the proportion of ALM traffic is very low, no more than 4%, which is acceptable for low programming time; 2) the node in a smaller region has fewer related routing rules, thus smaller region has lower ALM traffic ratio. Figure 12 shows the CDF of the FC table entries in typical regions. As for the storage overhead, with ALM, the average memory consumption for each vSwitch is 1,900 cache entries. The peak of the FC storage for a VPC with 1.5 million VMs is 3,700, which is much less than $O(N^2)$. We can find that ALM saves more than 95% memory usage and solves the routing table storage problem of hyperscale cloud networks with little extra overhead.

7.2 Elastic Network Capability

Effect of Elastic Credit Algorithm. To validate the effect of the elastic credit algorithm, we set up an experiment of the elastic network of VM1 and VM2 in the same host, as shown in Figures 13 and 14. We limit any of these two VMs' *base* bandwidth to 1000 Mbps. There are three stages: 1) in the first 30s, we use two VMs on the other host to send a stable flow at 300 Mbps to VM1 and VM2 respectively. The CPU consumption of both VMs is 20%; 2) After that, a bursty flow is sent to VM1 (30s to 60s). We observe that VM1 can briefly reach about 1500Mbps. Then, VM1 consumes all credits and is suppressed to 1000Mbps. The CPU consumption of VM1 reached 55% in a short time, and then fell back to 40%; 3) After 60 seconds, we send small packets to VM2, which will consume much more CPU resources and thus the CPU utilization rate of VM2 will reach 60%. The VM2 bandwidth can reach 1200 Mbps, which is then suppressed to 1000 Mbps as for the CPU-based elastic credit algorithm. Therefore, we observe that the vSwitch always strictly ensures the CPU resources allocated by VM1 to at least 40% (in case of resource contention), so it can ensure that the bandwidth of VM1 is basically unchanged. In practice, our BPS-Based+CPU-Based methods can also ensure latency due to eliminating resources competition, and 99% of the flows have latency within 300 μs. Since we deployed this mechanism, as shown in Figure 15, the average number of hosts suffering resources (CPU/Bandwidth) contention has decreased by 86%. In summary, our elastic credit algorithm with BPS-Based+CPU-Based methods can safeguard isolation and have high elastic performance in both simple and complex VPC scenarios.

Table 2: *Achelous* detected anomaly cases during the recent two months.

No.	Category	# of cases
1	Physical server CPU/memory exception.	12
2	Configuration faults after VM migration/release.	21
3	VM/Container network misconfiguration.	90
4	VM exceptions (memory/CPU exceptions, I/O hang).	12
5	The NICs have software exceptions or I/O hang.	45
6	VM hypervisor exception.	3
7	Middlebox CPU overload by heavy hitters.	15
8	vSwtich CPU overload by burst of traffic.	27
9	Physical switch bandwidth overload.	9
	In total	234

Effectiveness of distributed ECMP mechanism. As for the distributed ECMP mechanism, we deploy it in all the production cloud regions. With the seamless scale-out, we achieve the expansion and contraction of network services within 0.3s. Based on these technologies, we have accomplished that 80% of Alibaba Cloud network middleboxes (such as LB, NAT, VPN Gateway, etc.) have migrated to the VM on cloud as a form of Network Functions Virtualization (NFV). These middleboxes inside VMs have provided advanced network services for millions of tenants.

7.3 Effectiveness of Health Check and Live Migration

Anomaly Cases Detected by Health Check. As discussed in §6.1, *Achelous* can detect link failures and device failures. Through the warning of health check, *Achelous* can be aware of the possible failures in advance of tenants. Table 2 illustrates that *Achelous*

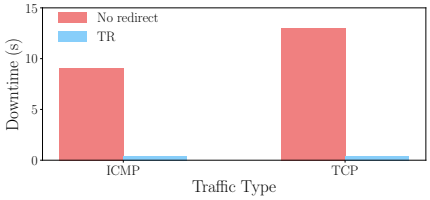


Figure 16: Effectiveness of TR.

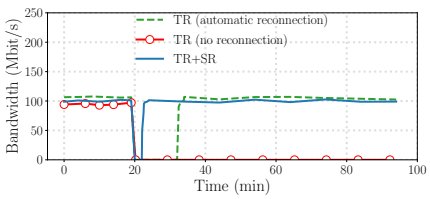


Figure 17: Effectiveness of TR+SR.

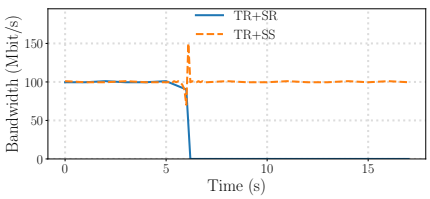


Figure 18: Advantage of TR +SS.

helps detect multiple classes of hardware failures (the first part of Table 2). Among them, CPU exception is the most common type as many virtual devices are based on CPU forwarding (*e.g.*, vSwitch, and gateway). Also, *Achelous* can detect resource contentions in the second part of Table 2, which may cause VM network jitter or performance degradation. Last but not least, *Achelous*'s health check mechanism is extensible. We can add more indicators to monitor in the future to quickly detect failure.

Downtime During Live Migration. We measure the downtime, the interval between migration and reconnection, under two circumstances (ICMP and TCP). Specifically, we first sequentially send the ICMP probe. We count the number of lost packets during migration so as to calculate the downtime. Second, we create TCP connections between VMs. We derive the downtime by checking the TCP seq number. Figure 16 shows the downtime of the TR scheme and the traditional no redirect scheme. We observe that TR can greatly reduce the downtime during live migration, and the downtime of TR is 400ms, which is 22.5× and 32.5× faster than the traditional method, under ICMP and TCP, respectively.

Effectiveness of Session Reset and Session Sync. In Figure 17, if an application has auto-reconnect function (see the green line), it will restart the application connection in 32s (default in Linux system). Otherwise (see the red line), the connection will be lost during the VM live migration. In contrast, our TR+SR only introduces 1s downtime. Therefore, our TR+SR can successfully reduce the waiting time before application reconnection. In scenarios such as the destination VM is configured with ACL rules, which only allow source VM in and reject any other VMs' traffic. As shown in Figure 18, we observe a blocked connection under TR+SR for lacking ACL rules in the new vSwitch, making the flow impossible to continue. In contrast, our TR+SS scheme enables the vSwitch to synchronize *session*; as such, the connection will not be blocked. It is worth noting that this scheme only introduces about 100*ms* of failure recovery latency. We conclude that *Achelous* live migration schemes are practical and low-latency, only consuming a few hundred milliseconds while guaranteeing the continuity of stateful flows.

8 EXPERIENCES

Achelous has been deployed in our hyperscale VPCs for years, improving the serviceability of the Alibaba cloud network. Even facing volatile production challenges such as increasing network scale, bursty traffic and malicious attacks, it still provides a great experience for tenants, so we believe we make the right things in the right places. Besides the designs in the paper, we have also accumulated lots of experience R&D for cloud networks. Now we would like to share them in this section.

8.1 Deployment Issues

Can the designs in *Achelous* be used in the hardware offloaded architectures? As the hardware offload method becomes a trend, major cloud vendors have all deployed the SmartNICs or CIPU-Based vSwitches (such as [18]) to improve VM network capacity. However, the fact is that non-CPU hardware, such as FPGA, cannot independently realize all functions of cloud vSwitch due to the lack of flexibility and iteration efficiency. So for *Achelous*, hardware plays the role of the accelerated cache (like the fast path mentioned in §2.3). Its implementation will not influence the collaborative designs in this paper. Moreover, Alibaba's experience in hardware offload acceleration in recent years also proves these co-designs can serve well on the SoC-based vSwitch.

Is *Achelous* ready to serve outside Alibaba Cloud? The main innovations on *Achelous* 2.0 are dedicated to addressing new challenges in hyperscale VPCs. It should be noted that none of these designs are implemented based on Alibaba Cloud's special hardware platform or software framework. This means that all these designs are based on simple principles, and can be easily implemented on any hardware and software platform. We believe that as the cloud computing market grows, other small and medium-sized cloud vendors will soon face the same scale issues like us, and they will benefit from our experiences.

8.2 Lessons Learned

Co-design is a trend for cloud network. As Moore's Law fails, it is more difficult to obtain good benefits from the over-design on a single module. The hardware offloading does not mean always providing stable high performance in complex scenarios. Therefore, the co-design between different network components is necessary for higher-quality virtual networks. Taking the example from the ALM mechanism (in §4.1), fast convergence in such a huge but variable network cannot be achieved without the cooperation of gateway, controller and vSwitch. We believe, in the future, the co-designs (including collaboration of both inter-components and hardware-software) will play a more critical role and go beyond what can be achieved by simply stacking resources.

Besides performance, the fast iteration and flexibility are also critical to network forwarding components. The iteration speed of forwarding components will directly determine how fast we can support new requirements from tenants and fix the existing bugs. These capabilities are critical to the evolution of cloud infrastructure. However, in implementation, we need to take into account both the flexibility and high performance in the architecture design. In *Achelous*, after several years of iteration, we adopt the design of decoupling service logic from acceleration paths. That enables us to realize the acceleration forms of software, hardware

and other different architectures under the same set of service logic. That's why the innovations in this paper are easily achieved in *Achelous* 2.0.

9 RELATED WORK

Network virtualization has been a research hotspot in the past decade. In terms of the data plane, the most popular open-source project Open vSwitch [43] first proposed the design of separating the fast path and the slow path. After that, a series of performance optimization methods, such as user-space component acceleration and hardware offloading schemes, are proposed [18, 19, 25, 51]. For the control plane, Google proposed B4 [26, 29] and Azure presented VFP [17] enriching SDN theory. However, as the development of cloud computing gradually enters a new stage, unilateral optimization is no longer to support the growing tenant requirements. Therefore, *Achelous* proposes the collaborative design of data plane and control plane to support hyperscsale VPC networks. We will briefly discuss the works most relevant to *Achelous*.

Hyperscale network programming. Pre-programmed model [31] is the first used to program software-defined network. However, its programming overhead increases quadratically with the size of the VPC, which is unacceptable for our ultra-large clusters. Andromeda [14] and Zeta [54] combined the advantages of the gateway model and the on-demand model. But the flow granularity they chose will make gateway a potential heavy hitter, and it is also more burdensome than *Achelous* because using a centralized node to decide offloading strategy.

Elasticity. There are many well-studied works on network bandwidth allocation, such as [8, 11, 22, 27, 32, 34, 45]. However, the traditional bandwidth policies ignore the consumption of multiple resources in the virtual network. Picnic [33] and authors in [7] mentioned that it is necessary to share and allocate network-dedicated CPU resources for tenants, but lack of unified system management. *Achelous* adopts a unified resource allocation algorithm for different resources, which not only ensures isolation but also fits network bursts. On the other hand, we noticed that there is no existing work on how to deploy ECMP routing in a virtual network. So we propose the distributed ECMP mechanism to overcome the bottleneck in centralized deployment and provide tenants the ability to scale out easily.

Reliability. For failure telemetry, there are lots of works [10, 20, 21, 23, 24, 47, 50, 56] dedicated to the failure telemetry in physical networks, but most of them cannot be directly used in virtual networks due to the variable virtual topology. So we develop a virtual network telemetry method in *Achelous* to perceive failure and do failover before the tenant awareness. For live migration, most of the existing works focus on host resource migration, *e.g.*, memory dirty pages on bare metal cloud [28], or SR-IOV network devices [53]. khai et al. [30] proposed a multi-step migration process to reduce downtime and ensure traffic continuity during migration, but cannot solve the continuity of stateful flows. In *Achelous*, the live migration with "TR+SS" mechanisms ensures the non-interruption of tenant's stateful services.

10 CONCLUSION

With the growing demand of cloud computing, cloud vendors need to host millions of instances in a single VPC. Thus it becomes extremely challenging for the fast network configuration convergence, high elasticity, and high reliability. To this end, we propose *Achelous*, which provides years of high *serviceability* for hyperscale VPCs in Alibaba Cloud. In *Achelous*, we show in detail that it achieves this goal with an ALM mechanism, elastic network capacity strategy, distributed ECMP mechanism, and network failure avoidance schemes. Compared with the existing well-studied designs on data plane, *Achelous* opens up a few exciting research directions regarding ensuring serviceability for cloud networks, which is the key success metric of cloud vendors in our opinion. *Achelous* does not rely on any custom hardware, so other small and medium-sized cloud vendors can benefit from our design and our deployment experience presented in this paper.

This work does not raise any ethical issues.

ACKNOWLEDGEMENT

We sincerely thank all the anonymous reviewers for their valuable feedback and constructive suggestions. This work was supported by the Fundamental Research Funds for the Central Universities 226-2022-00151, Alibaba Cloud through Alibaba Innovative Research Program, the Program of Zhejiang Province Science and Technology (2022C01044), the funding for Postdoctoral Scientific Research Projects in Zhejiang Province (ZJ2022072), the research was also supported by the advanced computing resources provided by the Supercomputing Center of Hangzhou City University.

REFERENCES

[1] 2022. A Detailed Explanation about Alibaba Cloud CIPU. https://www.alibabacloud.com/blog/599183?spm=a3c0i.23458820.2359477120.3.76806e9bESi3SD. (2022).

[2] 2022. Data Plane Development Kit (DPDK). https://www.dpdk.org/. (2022).

[3] 2022. Leaping ahead: The power of cloud network innovation. https://d1.awsstatic.com/events/Summits/reinvent2022/NET211-L_Leaping-ahead-The-power-of-cloud-network-innovation.pdf. (2022).

[4] 2022. Netfilter. https://www.netfilter.org/. (2022).

[5] 2023. Taobao homepage. https://www.taobao.com/. (2023).

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

[7] Vamsi Addanki, Leonardo Linguaglossa, James Roberts, and Dario Rossi. 2018. Controlling software router resource sharing by fair packet dropping. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 1–9.

[8] Max Alaluna, Nuno Neves, and Fernando M. V. Ramos. 2020. Elastic Network Virtualization. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE Press, 814–823. https://doi.org/10.1109/INFOCOM41043.2020.9155287

[9] Muhammad Khoirul Anwar and Ida Nurhaida. 2019. Implementasi Load Balancing Menggunakan Metode Equal Cost Multi Path (ECMP) Pada Interkoneksi Jaringan. *InComTech: Jurnal Telekomunikasi dan Komputer* 9, 1 (2019), 39–48.

[10] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically Finding the Cause of Packet Drops. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, 419–435.

[11] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. 2016. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, USA, 407–424.

[12] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live Migration of Virtual Machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design; Implementation - Volume 2 (NSDI'05)*. USENIX Association, USA, 273–286.

[13] Levente Csikor, Dinil Mon Divakaran, Min Suk Kang, Attila Kőrösi, Balázs Sonkoly, Dávid Haja, Dimitrios P. Pezaros, Stefan Schmid, and Gábor Rétvári. 2019. Tuple Space Explosion: A Denial-of-Service Attack against a Software Packet Classifier. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 292–304. https://doi.org/10.1145/3359989.3365431

[14] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc De Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, 373–387.

[15] 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.

[16] 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 (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 31–43. https://doi.org/10.1145/3387514.3405851

[17] Daniel Firestone. 2017. VFP: A Virtual Switch Platform for Host Sdn in the Public Cloud. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, USA, 315–328.

[18] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, 51–64.

[19] Peixuan Gao, Yang Xu, and H Jonathan Chao. 2021. OVS-CAB: Efficient rule-caching for Open vSwitch hardware offloading. *Computer Networks* 188 (2021), 107844.

[20] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI'19)*. USENIX Association, USA, 549–564.

[21] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 139–152.

[22] Jian Guo, Fangming Liu, John C. S. Lui, and Hai Jin. 2016. Fair Network Bandwidth Allocation in Iaas Datacenters via a Cooperative Game Approach. *IEEE/ACM Trans. Netw.* 24, 2 (apr 2016), 873–886. https://doi.org/10.1109/TNET.2015.2389270

[23] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 357–371.

[24] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, USA, 71–85.

[25] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. 2020. MASQ: RDMA for virtual private cloud. 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*. 1–14.

[26] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. 2018. B4 and after: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 74–87. https://doi.org/10.1145/3230543.3230545

[27] Shuihai Hu, Wei Bai, Kai Chen, Chen Tian, Ying Zhang, and Haitao Wu. 2016. Providing Bandwidth Guarantees, Work Conservation and Low Latency Simultaneously in the Cloud. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE Press, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524341

[28] Jaeseong Im, Jongyul Kim, Jonguk Kim, Seongwook Jin, and Seungryoul Maeng. 2017. On-Demand Virtualization for Live Migration in Bare Metal Cloud. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 378–389. https://doi.org/10.1145/3127479.3129254

[29] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined Wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/2486001.2486019

[30] Nguyen Tuan Khai, Andreas Baumgartner, and Thomas Bauschert. 2017. A multi-step model for migration and resource reallocation in virtualized network infrastructures. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 730–735.

[31] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. 2014. Network Virtualization in Multi-Tenant Datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, USA, 203–216.

[32] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. 2015. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/2785956.2787478

[33] Praveen Kumar, Nandita Dukkipati, Nathan Lewis, Yi Cui, Yaogong Wang, Chonggang Li, Valas Valancius, Jake Adriaens, Steve Gribble, Nate Foster, and Amin Vahdat. 2019. PicNIC: Predictable Virtualized NIC. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 351–366. https://doi.org/10.1145/3341302.3342093

[34] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. 2014. Application-Driven Bandwidth Guarantees in Datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 467–478. https://doi.org/10.1145/2619239.2626326

[35] Youngkon Lee and Ukhyun Lee. 2021. Reference Architecture and Operation Model for PPP (Public-Private-Partnership) Cloud. *Journal of Information Processing Systems* 17, 2 (2021), 284–296.

[36] Xianshang Lin, Yunfei Ma, Junshao Zhang, Yao Cui, Jing Li, Shi Bai, Ziyue Zhang, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. 2022. GSO-Simulcast: Global Stream Orchestration in Simulcast Video Conferencing Systems. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 826–839. https://doi.org/10.1145/3544216.3544228

[37] Ceh &Lt and C. Edu&Gt. 2000. *Analysis of an Equal-Cost Multi-Path Algorithm*. Technical Report 1. S265 pages.

[38] Chengfei Lv, Chaoyue Niu, Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, Tao Huang, Hui Shu, Jinde Song, Bin Zou, Peng Lan, Guohuan Xu, Fei Wu, Shaojie Tang, Fan Wu, and Guihai Chen. 2022. Walle: An End-to-End, General-Purpose, and Large-Scale Production System for Device-Cloud Collaborative Machine Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 249–265. https://www.usenix.org/conference/osdi22/presentation/lv

[39] Mallik Mahalingam, Dinesh 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*. Technical Report.

[40] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. 2022. ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 303–320.

[41] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. 2009. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*. Association for Computing Machinery, New York, NY, USA, 39–50. https://doi.org/10.1145/1592568.1592575

[42] 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, and Shunmin Zhu. 2021. Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 194–206. https://doi.org/10.1145/3452296.3472889

[43] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open VSwitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, USA, 117–130.

[44] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Bianchi. 2019. Flowblaze: Stateful Packet Processing in Hardware. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI'19)*. USENIX Association, USA, 531–547.

[45] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C. Mogul, Yoshio Turner, and Jose Renato Santos. 2013. ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 351–362. https://doi.org/10.1145/2486001.2486027

[46] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and K. K. Ramakrishnan. 2022. SPRIGHT: Extracting the Server from Serverless Computing! High-Performance EBPF-Based Event-Driven, Shared-Memory Processing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 780–794. https://doi.org/10.1145/3544216.3544259

[47] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. 2014. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 407–418.

[48] Luigi Rizzo. 2012. netmap: a novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*. 101–112.

[49] Vishal Shrivastav. 2022. Stateful Multi-Pipelined Programmable Switches. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 663–676. https://doi.org/10.1145/3544216.3544269

[50] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. Netbouncer: Active Device and Link Failure Localization in Data Center Networks. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI'19)*. USENIX Association, USA, 599–613.

[51] Janet Tseng, Ren Wang, James Tsai, Yipeng Wang, and Tsung-Yuan Charlie Tai. 2017. Accelerating Open VSwitch with Integrated GPU. In *Proceedings of the Workshop on Kernel-Bypass Networks (KBNets '17)*. Association for Computing Machinery, New York, NY, USA, 7–12. https://doi.org/10.1145/3098583.3098585

[52] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, et al. 2022. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 267–284.

[53] Xin Xu and Bhavesh Davda. 2016. SRVM: Hypervisor Support for Live Migration with Passthrough SR-IOV Network Devices. In *Proceedings of The12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '16)*. Association for Computing Machinery, New York, NY, USA, 65–77. https://doi.org/10.1145/2892242.2892256

[54] Qianyu Zhang, Gongming Zhao, Hongli Xu, Zhuolong Yu, Liguang Xie, Yangming Zhao, Chunming Qiao, Ying Xiong, and Liusheng Huang. 2022. Zeta: A Scalable and Robust East-West Communication Framework in Large-Scale Clouds. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1231–1248.

[55] Yi Zhao and Wenlong Huang. 2009. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. In *2009 Fifth International Joint Conference on INC, IMS and IDC*. IEEE, 170–175.

[56] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.

A DETAIL OF ELASTIC STRATEGY

Appendices are supporting material that has not been peer-reviewed.

The total bandwidth resources (CPU resources) for all VMs on the host are denoted by R^B_T (R^C_T). The actual bandwidth resource (CPU resources) usage of the VM is denoted by R^B_{vm} (R^C_{vm}). We set the default resource limit R^B_{base} and R^C_{base} for each VM. In addition, each VM has its own credit values $Credit^B$ and $Credit^C$. The VM can consume or accumulate its credit values. For example, if $R^B_{vm} < R^B_{base}$, in idle state, the vSwitch will accumulate the $Credit^B = Credit^B + (R^B_{base} - R^B_{vm})$ for the VM. If $R^B_{vm} > R^B_{base}$, in burst state, the vSwitch will consume the $Credit^B = Credit^B - (R^B_{vm} - R^B_{base})$ for the VM. Besides, the $Credit^B$ is bounded by $0 \leq Credit^B \leq Credit^B_{max}$.

When it comes to burst traffic, a VM can consume its credit values to use more resources. We also set resources maximum R^B_{max} and R^C_{max} for each VM, *i.e.*, $R^B_{base} \leq R^B_{vm} \leq R^B_{max}$ and $R^C_{base} \leq R^C_{vm} \leq R^C_{max}$. However, the host's resources may be insufficient for multiple VMs to consume their credits simultaneously. For example, if $\sum_{\forall} R^B_{vm} > \lambda \cdot R^B_T$, where λ is the threshold to measure whether the host is under resource competition, we will set R^B_τ instead of R^B_{max} for the Top-K heavy-traffic VMs ($\mathbb{T}_k$). Notice that $R^B_\tau \leq R^B_{max}$. We empirically adjust each VM's parameters, such as R^B_τ, R^B_{max}, and R^B_{base}. In case of extreme competition, all VMs are set to R^B_τ to guarantee the performance isolation, we ensure that the sum of VMs' R^B_τ are no more than R^B_T, *i.e.*, $\sum_{\forall} R^B_\tau \leq R^B_T$.

B DETAIL OF VMS LIVE MIGRATION

Traffic Redirect. When migrating stateless flows, the traditional method of VM migration is that the vSwitch learns the new path

through the gateway or control plane after the VM migration. Therefore, they will suffer from downtime in the order of seconds. To solve this issue, we propose TR to redirect the traffic to the vSwitch on the destination host.

Figure 9 shows the procedure of TR: After the controller sends the live migration command (including VM-host mapping) to the vSwitch2 on the source host, the VM2 migrates to the target host via the standard migration method ①, and the vSwitch2 issues a routing rule to route traffic to the VM2' on the target host. Then vSwitch2 acts as a routing node to redirect all the packets (vSwitch1 → VM2) to the vSwitch3 ②. Once redirect traffic hits the new rule, the vSwitch2 will send a reply packet to the vSwitch1. The TR will end until the vSwitch1 learns the new rules through ALM ③. As such, the VM successfully achieved live migration ⑦, and TR ensures the continuity of the traffic during live migration.

Session Reset. Next, we support the continuity of stateful flows. The stateful flows require each VM to save the flow state and session information, which has a tight dependency between the source and the destination VMs. Therefor, to provide a reliable live migration scheme and smoothly redirect stateful flows during migration, we should maintain the session/flow information during the migration process.

As shown in Figure 9, after the VM2' instance is established on the target host ①, the VM2 sends "reset" packets to the VM1 on HOST1 ⑤. The VM1 sends "syn" packets to VM2 (actually VM2') to establish a new connection ⑥. These "syn" packets and subsequent flows will be redirected by vSwitch2 to VM2' on the target host ②, so as to ensure the continuity of existing connections. Then, vSwitch1 learns new rules via ALM ③ and establishes a new connection directly with vSwitch3 ⑦.

Session Sync. However, the SR scheme needs to modify the customer's application to respond to TCP reconnection requests during the live migration, which reduces vendors' service compatibility and quality. We further propose Session Sync, only copying stateful flow-related and necessary sessions, to support the continuity of stateful flows from native applications which do not have reconnection ability or, we say, application unawareness.

As shown in Figure 9, after the VM2's new instance is established on HOST3 ①, vSwitch2 generalizes TR rules to redirect network traffic ②, the vSwitch3 will copy stateful flow-related and necessary sessions from vSwitch2 ④. In this way, the stateful links can continue to work as usual, and the on-demand copy will reduce the network damage rate by 50%. Finally, after vSwitch1 learns the new rules via ALM ③, it will establish a new connection directly with vSwitch3 ⑦.

Our live migration scheme only has milliseconds of downtime. So that *Achelous* can fast recover both stateless or stateful flows without awareness of native tenant services. Based on health monitoring and failure warning, we can smoothly migrate VMs to other hosts to avoid possible failures or quickly recover from failure, which greatly improves the reliability of the cloud network.