BIFROST: Alibaba's Next-Generation VPC Network with High-Performance Multipath Reliable Transport

Zihao Fan[†‡], Xing Li[§‡], Ye Yang[‡], Bo Jiang[†□], Bowen Yang[‡], Yilong Lv[‡], Yuke Hong[‡], Yinian Zhou[‡], Junnan Cai[‡], Jiayue Xu[‡], Yunrui Hu[‡], Zhao Gao[‡], Ke Sun[‡], Yimin Liu[‡], Xiangdong Zhang[‡], Enge Song[‡], Jianyuan Lu[‡], Xiaoqing Sun[‡], Shize Zhang[‡], Haonan Li[‡], Mingxin Li[‡], Changgang Zheng[‡], Yang Song[‡], Jun Liang[‡], Biao Lyu[‡], Rong Wen[¶‡], Zhigang Zong[‡], Shunmin Zhu[‡□]

[†]*Shanghai Jiao Tong University* [‡]*Alibaba Cloud* [§]*Zhejiang University* [¶]*Fudan University*

Abstract

Virtual Private Cloud (VPC) has become an increasingly important infrastructure-level service that provides isolated virtual networking environments for cloud tenants. In Alibaba Cloud, over 80% of tenant applications require reliable data transport, which is currently implemented in VPC networks through guest-side TCP. However, TCP often suffers significant performance degradation when facing network instabilities in production clouds, especially for tail-latency sensitive applications such as Redis and Nginx. In this paper, we present BIFROST*, the next-generation VPC network in Alibaba Cloud that provides high-performance multipath reliable transport.* BIFROST *employs RTT-aware multipath packet spraying to bypass failures and mitigate elephant flows, ensures in-order delivery via in-place guest reordering, achieves end-to-end reliability with delayed bitmap ACKs, and performs efficient reliability state management to support large-scale deployment. Extensive evaluation shows that* BIFROST *reduces tail latency by up to* 307× *for Redis and* 66× *for Nginx, achieves millisecond-level failure recovery, and supports O(100k) concurrent connections per SmartNIC.*

1 Introduction

Virtual Private Cloud (VPC) provides cloud tenants with a fundamental overlay network that enables isolated, self-managed networking environments. It has become an increasingly important infrastructure-level service for major cloud service providers (CSP) [11, 16, 32, 47]. In Alibaba Cloud, the number of VPCs has more than doubled from 3.2 trillion in 2020 to 8.0 trillion in 2024. Our statistics reveal that of all tenant application services over 80% depend on reliable transport (RT), and more than 50% are sensitive to tail latency [28]. Current VPC implementations rely on guest-side TCP for RT. However, our observation shows that TCP suffers significant performance degradation in the presence of network instabilities such as NIC flapping, elephant flows, and losses on virtual NICs (vNICs), which are not uncommon in production clouds. This is mainly because TCP uses single-path transmission and coarse-grained retransmission timeout (RTO).

Recent work has explored multipath transmission to improve the performance of RT. One line of work accomplishes this by customizing functions in programmable switches [1, 13, 31, 35–37, 58, 66, 68, 74, 75, 79]. Another line of work modifies the network stack, either on the guest side [22, 29, 54, 63, 82] or the host side [19, 21, 30, 36, 39, 43, 46, 59, 60, 65, 70, 80, 81]. Only a limited number of these studies explicitly address network virtualization and they invariably adopt the host-side approach [19, 46, 59, 60, 70]. In Alibaba Cloud, we implement multipath RT on the host side, because not all switches in the cloud are programmable, and modifying the guest-side network stack is intrusive for tenants. More specifically, our implementation is constrained to VPC's network virtualization layer, which serves as the interface between the virtual overlay network and underlying physical fabric. This strategic positioning has the additional advantage of facilitating loss detection across the entire datapath to enforce end-to-end reliability guarantees.

Implementing high-performance multipath RT in the host-side virtualization stack remains challenging, as evidenced by our experience of operating large-scale VPCs [40, 41, 76]:

- **Handling out-of-order packets.** Elephant flow mitigation typically requires packet spraying, which generates significant out-of-order (OOO) packet delivery at the host. Direct delivery of such OOO packets to the guest would trigger spurious retransmissions and significantly degrade the performance. In contrast, ensuring in-order delivery requires substantial buffering [65], typically in SRAM for the sake of throughput. Under a common production condition of 200Gbps bandwidth and 1ms RTT, this will require 25MB of memory. However, the virtualization stack is increasingly offloaded to SmartNICs, which provide only O(10MB) of SRAM to be shared by multiple critical packet-processing functions. The reorder buffer would create a substantial memory pressure, making in-order delivery challenging.
- **Reliable coverage.** In VPC networks, packet loss can occur

□Co-corresponding authors.

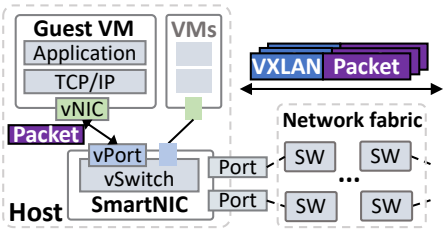

Figure 1: The basic architecture of VPC.

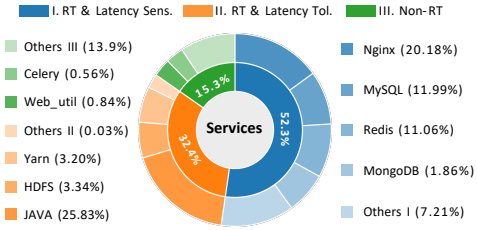

Figure 2: Service distribution.

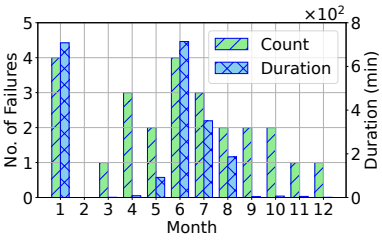

Figure 3: Failures in RT services.

on both physical links and virtualized in-host paths, in particular on vNICs. Existing host-side RT solutions generate ACKs at the physical NIC upon packet arrival, but this only covers the physical links and misses losses in virtualized paths [21, 39, 65]. Shifting the ACKs to the vNIC would solve the problem but requires guest kernel modification. Thus the challenge remains to achieve reliability that covers the entire datapath while staying transparent to tenants.

- **Scalability.** In our production environment, a single SmartNIC in the largest VPC already sustains up to O(100k) concurrent connections, making it challenging to maintain reliability states under the limited resources of SmartNICs.

To address these challenges, we propose BIFROST, the next-generation VPC network that enables high-performance multipath reliable transport. BIFROST employs RTT-aware multipath packet spraying to mitigate network instabilities (§4.1); achieves near-zero-buffer in-order delivery via in-place guest reordering, where the SmartNIC places OOO packets directly into guest memory and reorders only packet metadata (§4.2); ensures end-to-end reliability and fast loss recovery through bitmap-based delayed ACKs (§4.3); and scales to O(100k) connections by combining connection aggregation, sender-side delayed release, and resource pooling (§4.4). We have been developing BIFROST for a year with internal deployments in Alibaba Cloud (§5). Our evaluation shows that BIFROST reduces tail latency and sustains high throughput for critical services under various network conditions (§6). The insights from BIFROST can offer valuable guidance for the academic community and other cloud vendors (§7).

Our major contributions are as follows:

- We present extensive operational data from Alibaba Cloud, revealing performance challenges in production VPC networks and motivating our design of multipath RT.
- We propose BIFROST, the next-generation VPC network in Alibaba Cloud. It features RTT-aware multipath packet spraying, in-place guest reordering, bitmap-based delayed ACK, and efficient reliability state management.
- For deployment in VPC, we implement BIFROST with several techniques. BIFROST decouples control path and data path for SmartNIC offloading, preserves core affinity via flow redirect, and enables protocol version negotiation to ensure compatibility.
- BIFROST has undergone a year of development and internal deployment in Alibaba Cloud. BIFROST improves RT service performance, reducing tail latency by up to 307× for Redis and 66× for Nginx. BIFROST reduces memory consumption by 64.7%, supporting O(100k) connections per SmartNIC. We also share our lessons and insights.

2 Background and Motivation

In this section, we present the VPC architecture and describe the current state of RT services in the cloud (§2.1). We further discuss the network instabilities (§2.2), based on our monitoring data from large-scale deployment and operations.

2.1 VPC Architecture and RT Service

VPC architecture. VPC provides the foundation for tenant isolation in modern cloud infrastructure, often organized in a hierarchy across the VM, SmartNIC, and network fabric (Figure 1). Inside each VM, the guest OS runs a full network stack (*e.g.* TCP/IP) and sends/receives packets through a vNIC attached to a virtual port (vPort) of the virtual switch (vSwitch). The host, *i.e.,* the physical server, is equipped with a SmartNIC, in our case CIPU [25], an in-house SmartNIC developed by Alibaba Cloud. The CIPU executes core network virtualization logic and hosts the Apsara vSwitch (AVS) [40], which serves both as the endpoint of the physical network and the tunnel ingress for virtualized tenant networks. AVS supports tenant functions such as isolation [11], ACLs [5], and routing [6], and also handles VXLAN encapsulation for tenant traffic. It also provides monitoring tools such as Traffic Mirroring and Flowlog [3, 4, 14, 15], and adopts a hybrid hardware–software design for performance and flexibility [76].

RT service landscape. We present the distribution of tenant application services in Alibaba Cloud [28]. We classify the services as RT and non-RT according to whether they require reliable transport. As shown in Figure 2, RT services dominate the landscape, accounting for 84.72% of all services. Moreover, of all the RT services over 60% are sensitive to tail latency, including critical services such as Redis [56], Nginx [71], MySQL [51], and MongoDB [48]. Such services exhibit strict inter-packet dependencies, so even a single delayed packet could stall subsequent operations and cause severe performance degradation. For example, Redis, accounting for 11% of all services, exhibits ping-pong traffic with dependent requests, where tail latency directly amplifies end-to-end response time and degrades throughput. In contrast, other services such as HDFS [64] have weak inter-packet dependencies and are not blocked by delayed or lost packets, making them tolerant to tail latency.

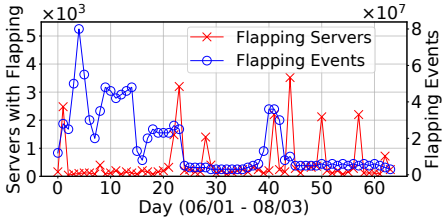

Figure 4: Physical NIC flapping.

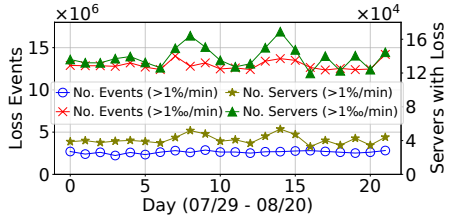

Figure 5: Packet loss in physical NICs.

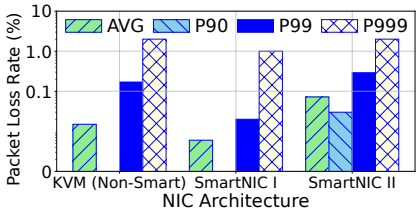

Figure 6: Packet loss in vNIC.

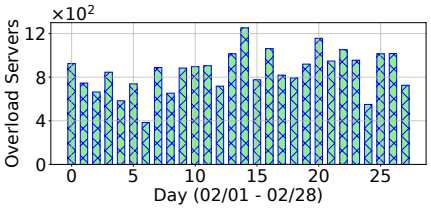

Figure 7: Servers with AVS overload.

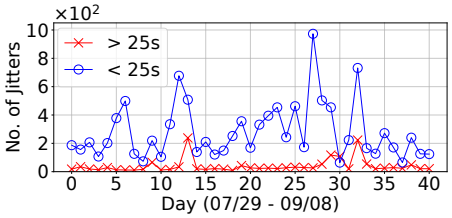

Figure 8: Physical network jitter.

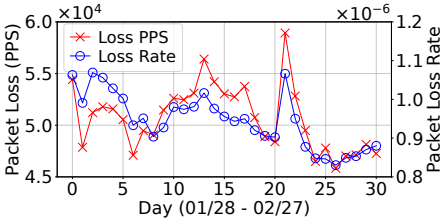

Figure 9: Packet loss in CGW.

Operational status of RT services. Currently, RT services rely on guest-side transport protocols such as TCP. However, due to its single-path nature, TCP is unable to handle network failures. Figure 3 shows the statistics of RT service disruptions due to network failures over a year. The services could experience up to four failures and more than 700 minutes of cumulative downtime monthly, causing substantial customer impact. Given the critical role of RT services, ensuring stronger reliability within the VPC is imperative.

2.2 Network Instabilities in Large-Scale Cloud

According to our experience, service disruptions typically stem from instabilities common in large-scale network infrastructure. We present below statistics from our monitoring data, with representative operational cases in Appendix B.

Host-side instabilities. In the production environment, a substantial fraction of reliability incidents originate within the host, where packets traverse multiple in-host segments, from physical NICs to the AVS, before reaching the tenant VM. Such instabilities can be categorized as follows.

Physical interface instability. NIC flapping (Figure 4) occurs when physical interfaces intermittently change link states, causing transient packet loss. From our two months of production monitoring across the entire cloud of O(1M) servers, we observe daily peaks of up to O(1k) affected servers and O(1M) flap events. Such disruptions force services to re-establish connections, adding latency and risking SLA violations. Beyond flapping, other factors such as bursty congestion also contribute to packet loss. To quantify their impact, we collect per-minute packet loss data from all physical NICs over a 20-day period (Figure 5). We define a loss event as any one-minute interval during which the packet loss rate exceeds 1%. Under this definition, O(10k) servers experience O(1M) loss events daily; at a 1‰ threshold, this expands to O(100k) servers with O(10M) events per day. These pervasive losses inflate tail latency and trigger retransmissions, driving SLA violations in latency-sensitive services.

In-host path congestion. Packet loss arises at a vNIC when its queues overflow, either because the guest drains the receive queue too slowly or the host-side AVS depletes the transmit queue too slowly. We compute the average packet loss rate over 12 hours for each vNIC in a region with O(100k) servers. Figure 6 shows the average over all vNICs and the 90th, 99th and 99.9th percentiles. Note that the 99.9th percentile exceeds 1% across different NIC architectures, making it essential to ensure vNIC reliability, especially for RT services.

> ***Takeaway 1:*** *Host-side instabilities motivate* ***(i) multipath transmission*** *to bypass flapping interfaces and* ***(ii) reliability coverage*** *across virtualized in-host paths.*

Network-side instabilities. The physical network datapath also experiences persistent, though less frequent, instabilities that nonetheless cause noticeable disruptions.

Elephant flows. It is well-known that a small fraction of long-lived, high-volume flows, *aka* elephant flows, can dominate link and CPU resources [44, 52, 69]. Hash collisions can place multiple elephant flows onto the same path, further aggravating congestion. We observe that elephant flows are not uncommon in cloud networks. Figure 7 presents the daily count of overloaded servers where the per-core utilization of AVS CPU ever exceeds 80%, over a one-month period in a region with O(100k) servers. We consistently observe *hundreds* of overloaded servers daily. As analyzed in [77], 7.8% of them experience elephant flows. These observations highlight the necessity of explicitly handling elephant flows.

Network jitter. In production networks, jitter-caused by transient congestion, dynamic switch buffers, and hardware pipeline delays-remains a persistent instability. As shown in Figure 8, 40 days of telemetry from the same region as in Figure 7 reveals *hundreds* of sub-25s and *dozens* of >25s jitter bursts per day, indicating that jitter is a recurring phenomenon rather than a rare anomaly. Such jitter could disrupt pacing, inflates queues, and triggers timeouts, ultimately degrading the performance of latency-sensitive services.

Takeaway 2: *Network-side instabilities necessitate* ***(i) packet spraying*** *to disperse elephant flows and* ***(ii) path-quality awareness*** *to avoid degraded paths under jitter.*

Middlebox instabilities. In scenarios such as Cloud Enterprise Network or inter-region connectivity [7], traffic often traverses multiple middleboxes, such as gateways, load balancers, and transit routers [8–10], whose limited capacity makes them prone to overload. When per-flow cores saturate, bottlenecks cause buffer overflows and packet drops. Figure 9 shows the rate and number of unplanned packet loss in Cloud Gateway (CGW) across the entire cloud over a one-month period. Although the average loss rate is below 0.01%, the massive traffic volume in CSP networks still results in O(10k) dropped packets per second (PPS). Such losses recover slowly under TCP's coarse-grained RTO, leading to inflated tail latency and degraded service performance.

Takeaway 3: *Packet losses from middlebox and host-side instabilities motivate* ***fast loss detection and recovery****.*

2.3 Potential Solutions and Their Issues

While numerous efforts have been made to address the above issues [1, 13, 19, 21, 22, 29–31, 35–37, 39, 43, 46, 54, 58–60, 63, 65, 66, 68, 70, 74, 75, 79–82], existing solutions still face limitations that hinder their practical adoption in VPC. We categorize the limitations into three key issues:

Limited performance improvement. Flow-level or flowlet-level multipath solutions [36, 37, 54, 74, 75] often struggle under elephant flows due to their coarse granularity. PLB [54] and LetFlow [74] rely on random path reselection, leading to slow convergence. MPTCP [29, 63] shows limited benefits in cloud settings due to VM-level bandwidth throttling and opaque vNIC-to-NIC mappings that prevent effective path aggregation [23]. SRD, Strack, REPS [21, 39, 59, 60] adopt packet spraying but lack reordering support at the receiver, incurring spurious retransmissions from OOO arrivals.

Intrusiveness to users. *Guest-side* solutions [22, 29, 54, 63, 82] like MPTCP and PLB require guest stack modifications, making them intrusive to deploy. Solutions like Clove, PLB and Flowbender [35, 36, 54] rely on ECN, typically disabled in tenant network stack [69], making them intrusive.

Poor compatibility and scalability. *Switch-side* solutions [1, 13, 31, 35–37, 58, 66, 68, 74, 75, 79] rely on customized switch functions for adaptive routing or in-network reordering, which are not universally available in large-scale networks. Moreover, these customized functions require frequent updates, which is impractical, as physical switch updates entail a large failure radius. In addition, switches are also a non-negligible of unreliability: our online failure recovery service ZooRoute [70] shows that 36.12% of intra-region link failures (2025/05/01–2025/09/01) stem from switch anomalies. In addition, some approaches [1, 20, 26, 53] depend on central servers, limiting their scalability for large-scale deployments.

2.4 Design Rationale and Challenges

Design goals. Building on the takeaways from §2.2 and the deficiencies identified in §2.3, achieving high-performance multipath RT requires three key objectives:

Path-quality-aware packet spraying. Traffic must be sprayed across multiple paths based on their quality to bypass failures and congestion, and disperse elephant flow hotspots.

In-order delivery. Packet spraying inevitably causes OOO arrivals; reordering must be completed before delivering to the guest to avoid spurious retransmissions.

End-to-end loss detection and fast recovery. Packet losses must be detected across the entire VPC datapath, including both physical and virtualized segments, and promptly recovered to minimize disruption.

Design choice. To realize these goals in VPC, the first question is where in the network stack to implement high-performance multipath RT. Existing designs fall into three categories: (i) *switch-side* customization, (ii) *guest-side* modification, and (iii) *host-side* enhancement. *Switch-side* designs require hardware upgrades with poor compatibility and scalability, while *guest-side* designs are intrusive to tenants. We therefore place our design on the host side. More specifically, We choose the host-side virtualization stack as the foundation for our design, which serves as both the endpoint of the physical network and the ingress of the virtual overlay. This dual role makes it a natural position for end-to-end reliability without tenant or physical switch changes.

Challenges. Realizing these goals in the host-side virtualization stack is non-trivial: the complex datapath in VPC and the limited resources of CIPUs raise three key challenges.

Handling OOO packets. Packet spraying improves robustness but induces OOO arrivals. Reordering them imposes substantial pressure on SmartNIC SRAM. The challenge is to reorder efficiently within the scarce SRAM of CIPU.

Reliability across the full VPC datapath. The challenge is to extend reliability beyond physical paths to virtualized in-host paths (*e.g.,* vNICs), while staying transparent to tenants.

Scalability under limited CIPU memory. Maintaining reliability state for O(100k) connections in production stresses limited CIPU memory [1], which requires carefully balancing between reliability and resource efficiency for scalability.

3 BIFROST Overview

In this section, we present the overview of BIFROST. We begin with an design overview of BIFROST (§3.1), followed by a brief description of its protocol stack (§3.2).

[1] In practice, even our most capable CIPUs have only 64 GB DDR memory shared by compute, storage, and networking hypervisors. Our measurements across O(10k) servers show some CIPU memory nearing saturation, with utilization at 34% (P99), 93% (P999), and 96% (P9999) [41].

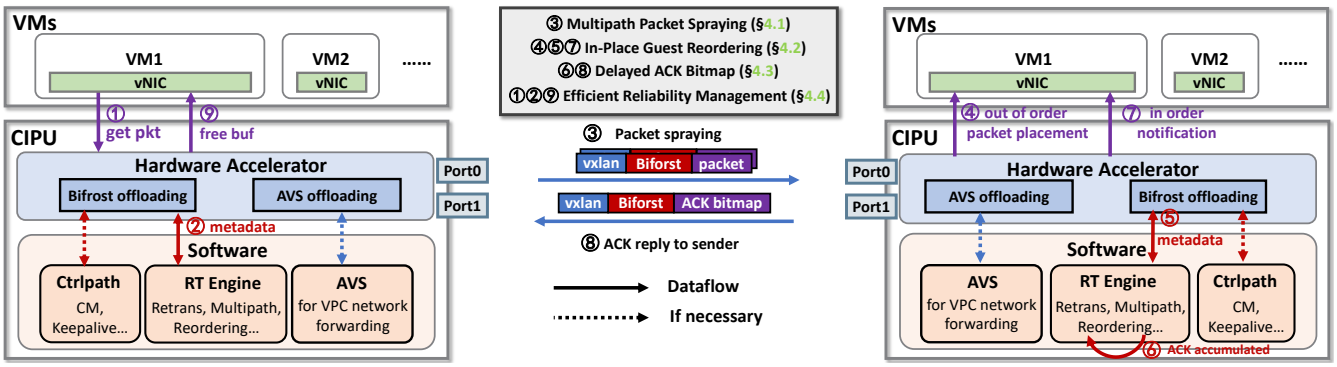

Figure 10: BIFROST overview.

3.1 Design Overview

Figure 10 illustrates the overview of BIFROST, which integrates several key components for high-performance multipath RT in VPC, including RTT-aware multipath packet spraying, in-place guest reordering, loss detection via a delayed ACK bitmap and efficient reliability state management. The four optimizations aim to address the challenges analyzed in §2.4. The main workflow of BIFROST is as follows.

When a VM sends a packet, it remains buffered in VM memory until acknowledged. The reliable transport engine (RTE) and AVS process the packet and spray it across multiple physical paths based on historical RTT measurements (§4.1). On the receiver side, OOO packets are placed directly into the guest VM memory of the destination VM without extra buffering on the CIPU. The RTE then reorders the associated packet metadata and issues an in-order notification to the VM (§4.2). Once a packet is transferred into guest VM memory, it is considered delivered. The RTE generates ACKs, which are aggregated into a bitmap and sent to the sender (§4.3). Upon receiving the ACK bitmap, the sender detects missing packets, identifies lost packets for retransmission, and releases the corresponding buffers for acknowledged packets (§4.4).

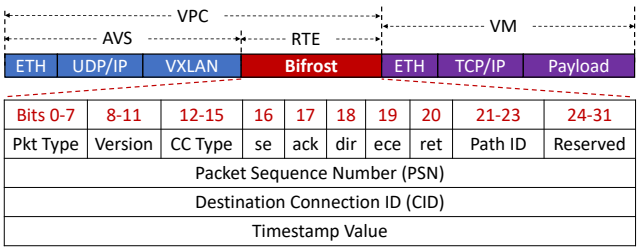

Figure 11: BIFROST protocol format.

3.2 BIFROST Protocol Stack

Figure 11 illustrates the protocol stack of BIFROST. Positioned above the VXLAN layer but beneath the tenant packet, BIFROST introduces a lightweight header that extends the virtualization boundary embedding RT semantics while remaining fully transparent to guest applications. The figure also shows the format of a data packet header. Several fields are particularly important for subsequent mechanisms: *Version* enables protocol version negotiation across heterogeneous NICs; the *Path ID* identifies paths within a multipath connection and enables packet-level steering; the *Packet Sequence Number* (PSN) provides fine-grained support for reordering at the receiver side; the *Connection ID* (CID) identifies a connection and is exchanged during connection establishment; and the *Timestamp Value* field enables precise RTT measurement for path quality estimation. These fields form the foundation of BIFROST's design and deployment. We provide their detailed specifications in Appendix C.

4 BIFROST Design

4.1 RTT-Aware Multipath Packet Spraying

To bypass failures and mitigate elephant flow hotspots, BIFROST uses RTT-aware multipath packet spraying. Packet-level spraying disperses elephant flows and avoids hash collisions; RTT-guided selection routes traffic to healthy paths.

Packet spraying strategy. BIFROST maintains an RTT table that records the latency and availability of each candidate path. Each flow is divided into small packet groups (*e.g.,* 50 packets per group), and BIFROST selects the *top-k* (*e.g.,* 4) paths with the lowest RTTs for transmission. These packet groups are then distributed across the selected paths in a round-robin fashion. For deterministic path selection, each physical path is mapped to a distinct *srcPort* in the outer five-tuple, and BIFROST rewrites this field to select the intended path.

RTT measurement. Timely RTT measurement is essential for tracking path quality. For active paths, BIFROST piggybacks RTT measurement on data traffic by extracting *timestamps* from ACK packets and updating the RTT table via exponential smoothing. By filtering out transient RTT spikes, BIFROST avoids unnecessary path switching and also quickly converges when path quality truly degrades. For idle paths that have seen no traffic for extended periods, the control plane periodically sends lightweight probes to maintain keepalive and refresh RTT. This dual strategy keeps BIFROST's path-quality view fresh on both active and idle paths, enabling fast failure detection and rerouting.

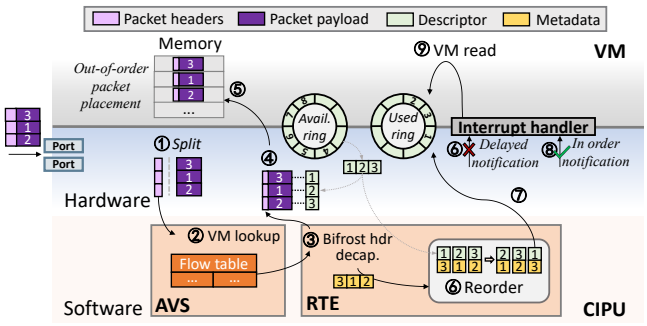

Figure 12: Workflow of in-place guest reordering.

4.2 In-Place Guest Reordering

To handle OOO packets from multipath packet spraying, we design an in-place guest reordering (IPGR) mechanism at the receiver side (Figure 12). A naive approach buffers all OOO packets on the SmartNIC hardware SRAM for reordering [65], which requires reorder buffers and could quickly exhaust the limited on-card resources. Our key insight builds on the de facto paravirtualized I/O interface (*virtio*) [57], which is ubiquitously deployed in large-scale clouds. In *virtio*, the VM consumes packets not in the order they are written into memory, but in the order their *descriptors* are written into the *used ring*. Leveraging *virtio's two-level mapping*, IPGR enables in-order delivery with near-zero buffering on the CIPU.

Out-of-order packet placement. Upon packet arrival, the CIPU decapsulates the VXLAN header and separates headers from payload for further processing (①). The headers are passed to the AVS to identify the destination VM (②) and then to the RTE for BIFROST header decapsulation (③). Instead of buffering the OOO packets on the CIPU, the CIPU fetches a descriptor from the VM's *available ring* (④) and DMA-transfers those packets into guest memory (⑤). As a result, packets thus are written in guest memory in the arrival order, while only their metadata (PSN and corresponding *descriptor*) is retained for subsequent reordering.

In-order notification to guest. To reconstruct in-order delivery, BIFROST RTE reorders only the metadata associated with each packet. After sorting, the CIPU writes the sorted *descriptors* into the *used ring* in the correct sequence and notifies the guest VM to consume the packets. From the guest's perspective, packets are consumed strictly in order from the *used ring*, even though they are placed in memory out of order. As illustrated in Figure 12, steps ④⑤⑥⑦ show this process: although packets are placed into guest memory in the arrival order $3 \rightarrow 1 \rightarrow 2$, the RTE reorders their metadata and writes the corresponding *descriptor* indices to the *used ring* in the sequence $2 \rightarrow 3 \rightarrow 1$.

Delayed interrupt for in-order notification. To ensure that the guest VM never observes OOO packets, BIFROST introduces a *delayed interrupt notification* mechanism. In the default *virtio* design, an interrupt is raised immediately after a packet is delivered into guest memory (*i.e.,* right after step ⑦ in Figure 12), which risks exposing OOO packets to the guest VM. To avoid this, BIFROST defers the interrupt until step ⑨, after reordered *descriptors* have been written into the *used ring*. This ensures the VM is interrupted only when in-order packets are ready for consumption.

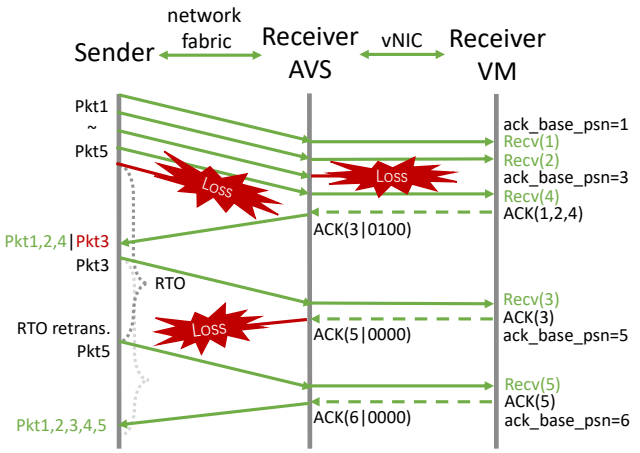

Figure 13: Loss detection via delayed ACK bitmap.

Overall, IPGR achieves in-order delivery with near-zero buffering. Only metadata is buffered on the CIPU, while packets are placed into guest memory immediately using the standard *virtio* buffers, incurring no extra memory burden to the VM. The temporary footprint on guest memory is negligible (KBs vs. the tens of GBs typically provisioned for VMs). The performance impact is also minimal, since only headers and metadata are exchanged between hardware and software in CIPU during reordering, rather than full packets. Finally, IPGR is implemented entirely on the CIPU side, requiring no guest changes and remaining transparent to tenants. Note that recent Linux kernels already support TCP RACK-TLP [24], which enhances tolerance to OOO packets; we discuss our reordering choices under such kernels in §7.3.

4.3 Loss Detection via Delayed ACK Bitmap

ACK aggregation via delayed bitmap. We refine the ACK mechanism with two techniques for accurate and efficient delivery confirmation. First, to extend reliability coverage across the entire VPC datapath, including virtualized segments (*e.g.,* vNICs), we introduce the *delayed ACK*. Unlike solutions that acknowledge a packet once it reaches the host protocol stack [21, 39, 65], BIFROST defers ACK generation until the packet has been successfully written into the guest RX queue, extends reliability coverage to the guest. Second, to efficiently track the delivery status of multiple packets, we employ a *bitmap-based ACK*, which aggregates the delivery status of multiple packets into a bitmap. The sender quickly infers received and missing packets from the bitmap.

Fast loss detection and recovery. To achieve fast loss detection and recovery, BIFROST combines bitmap-based gap detection with low-latency RTO. The ACK format `ACK(`a`|xxxx)` confirms that all packets with PSN $< a$ have

been received, while the bit string specifies the reception status of subsequent packets starting from *a*. Upon receiving an ACK bitmap, the sender detects discontinuities in the bit string and immediately triggers fast retransmission. For tail losses, where no discontinuity appears in the bitmap, and for cases where ACKs are lost, BIFROST relies on a low-latency RTO. The initial RTO starts at 4 ms (roughly 2 RTTs) and grows exponentially for subsequent retransmission of the same packet (*e.g.,* 8 ms, 16 ms, ...), ensuring timely loss recovery.

We illustrate this process in Figure 13. When the sender transmits packets `1-5` and the receiver VM receives only `1, 2, 4`, the AVS generates an `ACK(3|0100)`. This bitmap means that all packets with PSNs less than `3` have been received. Starting from PSN `3`, the bitmap `0100` indicates that the packet with PSN `4` is received, but packets with PSN `3` is lost. BIFROST triggers fast retransmission of packet `3`. When the receiver VM later receives packet `3`, all packets with PSNs below `5` have been received, and the AVS generates `ACK(5|0000)` to return to the sender. However, this ACK is lost during transmission. At the same time, packet `5` expires on the sender due to RTO and is retransmitted. When the VM receives the retransmitted packet `5`, the AVS generates `ACK(6|0000)` and sends it to the sender. After receiving the ACK, the sender confirms that packets `1-5` have been successfully delivered, thus finalizing transmission.

4.4 Efficient Reliability State Management

RT requires maintaining per-connection state as well as packet caches for potential retransmissions. When scaled to millions of tenants and highly concurrent connections, these requirements impose heavy CIPU memory pressure and limit scalability. To mitigate this, BIFROST applies a set of resource allocation optimizations that reduce memory overhead.

Connection aggregation. Within a VM, a transport connection is fundamentally established at the 5-tuple granularity, uniquely identified by the combination of *protocol*, *IPs*, and *ports*. Mapping every 5-tuple to a dedicated BIFROST connection on the CIPU would create an excessive number of active connections, quickly exhausting CIPU memory and limiting scalability. To address this, BIFROST introduces a vNIC-level aggregation mechanism: instead of maintaining per-5-tuple connection, all flows between the same pair of source and destination vNICs are multiplexed onto a single logical BIFROST connection. However, such aggregation introduces potential HoL blocking across multiplexed flows. This creates a trade-off between scalability and isolation. Through careful balancing, we choose the vNIC-pair granularity, which achieves substantial scalability gains while confining HoL impact within well-defined boundaries, as detailed in §7.2.

Sender-side delayed release. To ensure RT, the sender must temporarily retain unacknowledged packets to enable potential retransmissions. A naive design would buffer the full packets on the CIPU until corresponding ACKs arrive. However, since the tenant VM's TCP stack already keeps a copy of each outstanding packet, duplicating it on the CIPU leads to memory waste and poor scalability. To eliminate this redundancy, BIFROST adopts a sender-side delayed release mechanism: the CIPU stores only lightweight metadata, including the packet *descriptor*, instead of full packets, while the packets remain pinned in the VM. This means the *descriptor* is retained on the CIPU after transmission, rather than immediately recycled into the *used ring*. If retransmission is needed, the CIPU uses the *descriptor* to fetch the packet from VM memory and resend it. Once the ACK is received, the CIPU releases the *descriptor* to the *used ring*, allowing the VM to reclaim the buffer. Note that this mechanism requires modifications to the SmartNIC hardware processing logic, as the default workflow would immediately recycle *descriptors* after transmission. To better understand this mechanism, we provide the detailed workflow in Appendix E.

Resource pooling. Beyond packet buffering, RT also consumes substantial state per connection. A simplistic approach is to statically reserve these transmission resources for every connection, regardless of activity. At cloud scale, this static reservation quickly exhausts CIPU memory, as many connections may remain idle at any given time. BIFROST instead adopts a demand-driven pooling mechanism that allocates resources only for *active* connections. Per-connection structures, such as queues and TX/RX windows, are backed by a global pool. The per-packet resources within these structures (*e.g.,* metadata entries) are allocated on demand upon packet arrival, while idle connections consume only minimal memory. For example, a static reservation allocates 256 metadata entries (16 bytes each) per connection. In contrast, BIFROST replaces this with dynamic allocation from a shared metadata pool, provisioning entries only when needed.

5 BIFROST Implementation

5.1 Deployment Issues

There are a few deployment issues to be addressed.

Offloading BIFROST to CIPU. The CIPU typically consists of a hardware accelerator (*e.g.,* FPGA or ASIC) for datapath processing with an embedded SoC (typically ARM cores) for control-plane software. This architecture forms the foundation of network virtualization in Alibaba Cloud, supporting functions such as the AVS fastpath/slowpath for high-performance packet processing [40]. When offloading BIFROST to the CIPU, we face three major challenges: (1) partitioning functionality between the hardware datapath and the SoC control plane to balance efficiency and flexibility. (2) exploiting multi-core performance on the SoC while preserving per-connection consistency. (3) seamlessly integrating the BIFROST RT logic with existing AVS functions.

Adaptation to heterogeneous NICs. In large-scale production clouds, server fleets are highly heterogeneous: even within a single region, servers may use different generations of CIPUs or legacy commercial NICs with no programmabil-

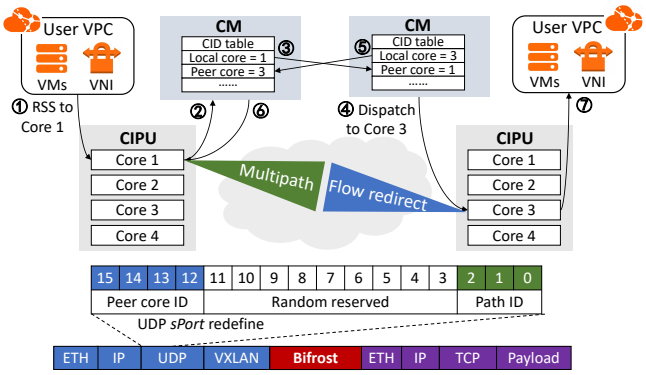

Figure 14: Workflow of flow redirect.

ity. To enable cloud-wide deployment, BIFROST must operate correctly across this diverse hardware landscape and gracefully adapt to varying levels of offload capability.

5.2 Offloading BIFROST to CIPU

Separation of datapath and control path. In BIFROST, the datapath and control path (ctrlpath) are explicitly separated to balance performance and flexibility. The hardware accelerator of the CIPU (*e.g.,* FPGA) is dedicated to datapath operations such as I/O, DMA, and high-speed packet forwarding. All RT-related modules in the datapath, including multipath scheduling, reordering, ACK processing, loss detection, and retransmission, are implemented in software on the embedded SoC, together with ctrlpath tasks such as connection management (CM) for flexibility. The hardware accelerator and the SoC exchange only lightweight packet metadata, minimizing PCIe traffic to support high-performance transport. Moreover, in ASIC-based architectures with a Data Process Accelerator (DPA) [50], which integrates a set of embedded RISC-V cores, the RTE modules can run directly on the DPA. In this case, the hardware acceleration goes beyond basic I/O and DMA, allowing the entire datapath to be offloaded, while the ctrlpath remains on ARM cores for programmability.

Multi-core offload coordination. Packet processing on the CIPU naturally runs across multiple cores to scale with increasing workloads. The challenge, however, is to preserve connection consistency: *each connection must be bound to a single core on both the sender and receiver sides for all data and ACK packets.* Any violation of this affinity undermines transport consistency and incurs costly cross-core synchronization. To address this, BIFROST establishes end-to-end core affinity at connection setup through a lightweight *flow redirect* mechanism (Figure 14). When a new connection is initiated, for example from Core 1 on CIPU 1 (①), the local CM records this core ID and triggers connection establishment (②). The sender then encodes the UDP *sPort* of its local core into the connection request (③). At the receiver, the CM extracts and stores the core ID of the sender, then selects a least-loaded local core to handle the connection (④), and encodes this core ID into the reply's *sPort* (⑤). Finally, the sender extracts the peer core ID from the response and stores it in the CM for future packet steering (⑥).

AVS cooperation and packet pipeline. The BIFROST RTE is deployed as a decoupled module that cooperates with AVS. We next describe the BIFROST packet pipeline to illustrate how it cooperates with AVS's existing architecture.

Connection establishment. When AVS receives a packet from a local VM, it parses the packet and extracts key metadata (*e.g., vPort*) to establish a vNIC-pair–level connection. Based on this information, AVS maps the packet to a CID and performs a lookup in the local CM. If no matching CID is found, BIFROST's ctrlpath initiates the connection setup procedure. The ctrlpath allocates a new CID entry and sends a connection request to the remote server. The remote server performs a similar allocation and replies with a connection response that includes its own CID and other required metadata (*e.g.,* core ID for flow redirect). After both endpoints exchange their CIDs, an RT connection is established.

Packet transmission. After connection setup, we describe the end-to-end packet workflow along the TX and RX paths, where BIFROST collaborates with AVS for delivery.

TX workflow. Once a packet matches an existing CID entry in the CM, its metadata is first used by the BIFROST RTE to perform transport-level operations, including multipath selection, timer initialization, and BIFROST encapsulation. After the RTE finishes processing, the packet metadata is returned to the AVS, which then performs network functions such as ACL, routing, and VXLAN encapsulation. Then, the AVS transmits the encapsulated packet onto the physical network.

RX workflow. Upon receiving a packet, the AVS first decapsulates the VXLAN header and then RTE parses the BIFROST header to determine whether it is an ACK or a data packet. If it is an ACK packet, the BIFROST RTE parses the bitmap to identify successfully delivered packets and release their corresponding resources. Any missing packets are immediately scheduled for fast retransmission. If it is a data packet, it is directly placed into guest memory. The IPGR (§4.2) guarantees in-order delivery and notifies the VM only when packets are ready after reordering. The BIFROST RTE then generates the corresponding ACK. Once the accumulated ACKs satisfy the bitmap threshold or no subsequent packets arrive within a timeout, an ACK bitmap is sent back to the sender.

5.3 Adaptation to Heterogeneous NICs

To achieve uniform deployment across heterogeneous hardware, BIFROST incorporates both host-based fallback and protocol-level mechanisms to ensure compatibility.

Non-programmable NICs. For non-programmable NICs, BIFROST falls back to a host-based execution path. In this setting, AVS runs on the host using DPDK [27] for packet I/O, and BIFROST provides its reliability functions in the same pipeline. This fallback requires no additional hardware support while preserving basic correctness and end-to-end reliability, enabling uniform deployment even on legacy servers.

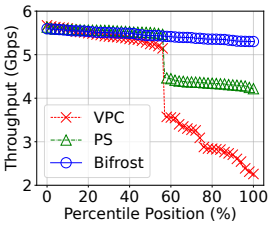

Figure 15: Flow throughput percentiles.

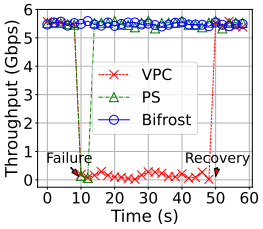

Figure 16: Impact of port failure on throughput.

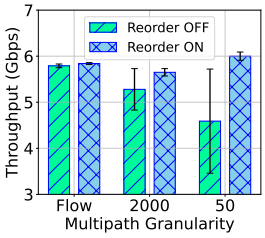

Figure 17: Thpt vs. granularity w/wo IPGR.

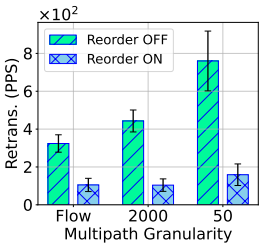

Figure 18: Retrans. vs. granularity w/wo IPGR.

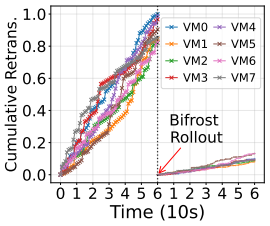

Figure 19: Cumulative retransmissions.

Protocol compatibility across heterogeneous NICs. Heterogeneous NICs offer asymmetric feature sets, which makes protocol negotiation necessary to avoid capability conflicts. For example, sender-side delayed release cannot be applied when reliability is offloaded to commodity DPUs (*e.g.,* NVIDIA BlueField3 [49], Intel IPU [34]), since delaying the update of the *used ring* requires customized hardware support. To avoid such conflicts, BIFROST introduces protocol version negotiation during connection setup: both sides exchange their protocol *version* field in the header, and the connection is established with the maximal common feature set supported by both versions. This ensures compatibility across heterogeneous NICs without assuming specific hardware features.

6 Evaluation

6.1 Experimental Settings

Testbed setup. We first set up a testbed to evaluate each component of BIFROST. The testbed consists of dozens of servers, each equipped with a CIPU powered by a CPU-FPGA architecture, whose SoC has a 16-core Intel Xeon CPU@2.0 GHz and 64 GB memory. The client and server VMs are placed on two different physical servers, each VM equipped with a 128-core Intel Xeon CPU@2.90GHz and 256 GB memory. The results in §6.2 are from the testbed.

Cluster deployment. We then deploy BIFROST on Alibaba Cloud's test clusters to evaluate its performance. These clusters closely resemble production environments and are inherently multi-tenant with concurrent tasks that introduce realistic interference. Within this setting, we evaluate tail-latency-sensitive RT services to capture their end-to-end performance under realistic cloud conditions. The results in §6.3 are collected from test clusters.

6.2 Microbenchmarks

6.2.1 Multipath Packet Spraying

We evaluate the effectiveness of *RTT-aware multipath packet spraying* in two representative scenarios: elephant flows and failure recovery. In both cases, we compare BIFROST against Probe-and-Switch (PS) and the vanilla VPC. PS is a path-switching approach that relies solely on probe packets to detect and migrate to alternative paths, with path probing performed at second-level intervals. The vanilla VPC relies on guest TCP, which by default uses single-path transmission.

Handling elephant flows. To evaluate how BIFROST handles concurrent elephant flows, we run experiments where a client VM initiates two long-lived flows to a single server VM using *iperf*, repeating this process 100 times. On the receiver side, each NIC port is rate-limited to 3 Gbps to create a contention hotspot. For each test, we measure the bandwidth over a 5-second interval and calculate the average. We record the aggregate throughput of the two flows for each run and plot its percentile distribution for each method (Figure 15). The vanilla VPC performs poorly when two flows hash to the same port due to the lack of migration, while PS mitigates this but still suffers high-percentile throughput drops from its seconds-level convergence delay. In contrast, BIFROST maintains a consistently high aggregate throughput across all percentiles by promptly dispersing packets across multiple paths based on RTT feedback, avoiding the prolonged collisions that degrade the performance of VPC and PS.

Failure recovery. To evaluate how BIFROST responds to path failures, we also use *iperf* to generate a long-lived flow from a client to a server. We record the throughput every second over a 60-second period. At the 10-second mark, we introduce artificial congestion by injecting a 10% packet loss rate on the active physical port to simulate a failure scenario, and restore the port to zero loss at the 50-second mark. As shown in Figure 16, BIFROST maintains stable bandwidth during the failure, rapidly shifting traffic away from the degraded port to sustain performance. In comparison, VPC suffers a severe bandwidth drop and fails to recover until the port is restored, while PS achieves path switching within a few seconds but still experiences significant throughput degradation. This is because PS's second-level probing limits failure convergence time to several seconds, and VPC lacks any multipath capability to reroute around failures. In contrast, BIFROST piggybacks RTT measurements on in-flight packets, enabling rapid path quality detection and adaptive traffic redistribution.

6.2.2 Packet Reordering

We evaluate the impact of IPGR (§4.2) on performance under different multipath granularities, including flow-level, 2000-packet, and 50-packet switching. Using *iperf*, a client VM transmits a long-lived flow to a server VM under each setting. We measure throughput and VM's retransmissions with reordering enabled and disabled, as shown in Figures 17 and 18. Our results indicate that enabling IPGR consistently

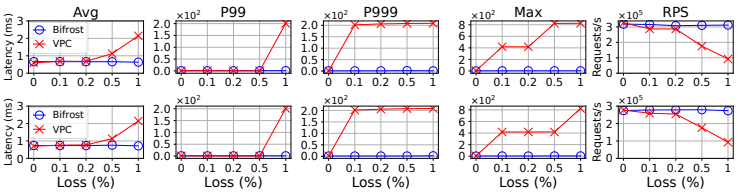

Figure 20: Redis performance (top: get, bottom: set).

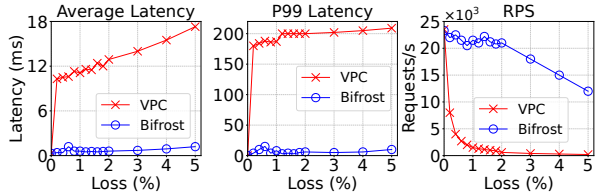

Figure 21: Nginx performance.

improves throughput by 1.01×–1.31× and reduces retransmission count by 3.09×–4.78× across all granularities. When reordering is disabled, finer-grained multipath results in increasingly severe throughput degradation and elevated retransmissions, as frequent path changes cause OOO arrivals that trigger spurious loss recovery in the VM's transport stack. IPGR consistently maintains high throughput and low retransmission rates across all granularities by reordering OOO packets before they reach the transport stack, which shields the transport stack from spurious loss recovery, even under highly disordered packet arrivals.

6.2.3 Reliability Coverage

To verify BIFROST's end-to-end reliability across the entire VPC datapath, we inject packet loss on the receiver side between the AVS and the vNIC, where in-host drops frequently occur. We conduct an 8-to-1 incast experiment using *iperf*, where eight client VMs on separate physical servers concurrently transmit long-lived flows to a single server VM. The injected loss rate is dynamically varied every second within a random range of 0% to 2%, and the test runs for 120 seconds. At the 60-second mark, we enable BIFROST, and we measure the cumulative retransmissions for the periods both before and after activation. As shown in Figure 19, BIFROST effectively handles in-host packet loss and prevents it from escalating to the VM, thus reducing VM retransmissions by 69.1% to 90.0%. This demonstrates that the delayed ACK mechanism provides end-to-end reliability and offloads most reliability responsibilities from the guest kernel to BIFROST.

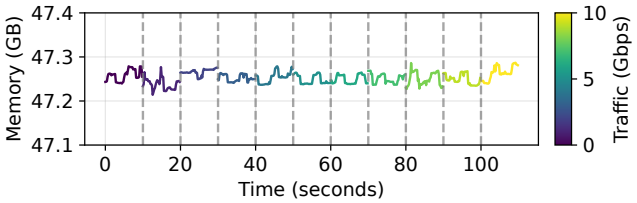

Figure 22: Memory usage on the sender-side CIPU.

6.2.4 Resource Optimization

Connection aggregation. To evaluate the benefit of connection aggregation in realistic settings, we collect connection statistics from production servers, where each server hosts multiple tenant VMs. For each server, we count the number of active connections at the 5-tuple granularity and compare it with the number of connections after vNIC-pair-level reuse. Across hundreds of servers, connection aggregation reduces BIFROST connections by 70% on average, validating its effectiveness in production multi-tenant environments.

Sender-side delayed release. Since sender-side delayed release is hardware-dependent and not runtime-switchable, we cannot directly toggle it in experiments. To isolate its benefit, we run a single BIFROST connection between a client and a server and measure the sender's CIPU memory growth under varying traffic volumes, where the gains from connection aggregation and resource pooling are negligible. Figure 22 reports the sender-side CIPU memory usage, starting with BIFROST disabled at 0 Gbps and then enabled as traffic increases from 1 Gbps to 10 Gbps in 1 Gbps steps every 10s. The overhead is minimal: buffering 16 bytes of metadata per packet yields only ∼500 KB at 10 Gbp, which is negligible compared to GB-scale requirements of full-packet caching.

Resource pooling. We evaluate how the resource pooling reduces per-connection memory overhead and improves scalability. To measure the per-connection state size, we establish a controlled number of connections from the client VM and monitor the incremental memory consumption on the sender-side CIPU. By comparing the total memory increase before and after enabling resource pooling, we compute the average state footprint per connection. Our measurements show that resource pooling reduces the per-connection state size from 34 KB to 12 KB, which improves connection scalability under the same memory budget. At this footprint, supporting $O(100\text{k})$ connections requires $O(1)$ GB of CIPU memory, which remains within a practical range for deployment.

6.3 Application Performance

Redis. We evaluate Redis, a widely used tail-latency-sensitive tenant service in Alibaba Cloud. The experiments use two VMs on different physical servers in test cluster, running *redis-benchmark* with persistent connections to test latency and throughput. To emulate packet loss, we configure the receiver-side AVS to drop packets at controlled rates from 0% to 1%. Figure 20 reports the measured latency and throughput under different loss rates. BIFROST reduces *get* latency by up to 3.4× (Avg), 242.1× (P99), 210.5× (P999), 306.9× (Max), while improving throughput by 3.4×; for *set*, the gains are up to 2.9×, 196.7×, 182.4×, 230.1×, and 2.9×, respectively. Without BIFROST, tail latency inflates sharply once loss exceeds 0.1%, with P999 and Max rising above 200 ms. At 1% loss for *get*, throughput collapses from 327k to 92k requests per second (RPS). In contrast, BIFROST keeps tail latency below 4 ms and sustains around 300k RPS under 1% loss.

These improvements stem from bitmap-based loss detection and low-latency RTO, enabling fast recovery.

Nginx. We also evaluate Nginx short-connection mode under various loss rates. Its performance mirrors that of Redis. The average latency is reduced by up to 26.3×(20.2× on average), P99 latency is lowered by up to 66.7× (38.3× on average), and throughput (RPS) improves by up to 60.0× (23.9× on average). Figure 21 illustrates this stark contrast. As loss increases, the average and P99 latency in the vanilla VPC rise sharply, driving throughput down to just a few hundred RPS. In contrast, with BIFROST, both average and P99 latency remain at a consistently low, under 10 ms. Throughput is also sustained at 20k RPS even under 2% loss, mitigating the severe degradation observed in the vanilla VPC.

7 Experience

7.1 Principles of Handling Packet Loss

While our design primarily targets packet losses caused by network instabilities, we observe that in production environments, a significant portion of losses stem from tenant policy configurations, such as ACL [5] or bandwidth rate limiting. These losses differ fundamentally in nature and require distinct handling strategies. We therefore define two categories of loss and establish separate handling principles:

Losses from network instabilities. These are caused by instabilities on physical or virtual paths, as described in §2.2. Since these losses are unexpected, BIFROST fully takes over the retransmission responsibility, ensuring timely recovery.

Losses from policy enforcement. These intentional losses from policy enforcement fall into two cases.

ACL and security group drops. In this case, these drops are caused by access control policies that silently discard packets at the receiver side. We intentionally make BIFROST ignore such drops, and the receiver should therefore not trigger retransmissions. To maintain end-to-end correctness, BIFROST generates synthetic ACKs for the discarded packets so that the sender can advance its window without stalling.

Rate-limiting drops. When packet drops are caused by bandwidth enforcement policies, BIFROST must detect and react accordingly. Specifically, the receiver marks the dropped packets in the ACK bitmap. Upon receiving this feedback, the sender naturally reduces its congestion window following our RTT-based congestion-control logic, which effectively aligns its sending rate with the enforced bandwidth policy.

7.2 Handling HoL Blocking Issue

Balancing scalability and isolation. Any connection aggregation scheme incurs the risk of HoL blocking, especially when multiple flows are multiplexed over the same logical connection. The granularity thus presents a trade-off: coarser levels save more state but increase interference risk. Aggregating at the VM-pair level would merge traffic from different containers, causing intra-VM contention, while AVS-pair aggregation would mix traffic across tenants, compromising isolation. We adopt vNIC-pair-level aggregation, which reduces BIFROST connections by over 70% compared to per-5-tuple states. This captures most memory savings while confining HoL blocking to well-defined endpoints such as container or tenant interfaces, preserving both scalability and isolation.

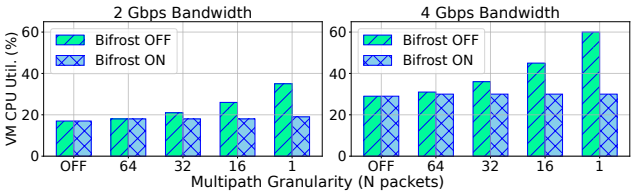

Figure 23: VM CPU util. reduction from BIFROST offload.

Mitigating residual HoL effects. Within a single vNIC, although HoL blocking can theoretically occur when one flow is stalled (*e.g.,* due to loss or congestion), BIFROST mitigates this issue via fast loss recovery, minimizing its impact. In extreme cases where all available paths fail and HoL persists, BIFROST falls back to the original VPC logic, reverting to best-effort UDP delivery to ensure basic connectivity.

7.3 Reorder Tradeoff Across Kernel Versions

Based on our deployment experience, we observe that the necessity of reordering offload heavily depends on the guest kernel version. In recent Linux kernels that support TCP RACK-TLP [24], the transport stack is more tolerant to OOO packets. In these environments, we leave it to the user to decide whether to enable the reordering in BIFROST. When reordering is disabled, guest TCP handles OOO arrivals, which reduces memory and processing pressure on the CIPU and allows support for a larger number of concurrent connections.

However, under severe reordering, common in fine-grained multipath scenarios, we find that guest-side reordering can consume substantial guest CPU resources, degrading tenant application performance. To quantify this effect, we conduct an experiment by varying the multipath granularity and measuring the receiver guest VM CPU utilization. As shown in Figure 23, finer-grained packet spraying leads to more severe packet reordering, which increases VM CPU overhead when reordering is handled in the guest kernel. In contrast, enabling BIFROST's reordering reduces VM CPU consumption by offloading reordering to the CIPU. In addition, when BIFROST reordering is enabled on kernels with RACK-TLP support, we automatically disable guest-side reordering logic by setting `tcp_recovery=0` and lowering `tcp_max_reordering`, so that all reordering is fully handled in the CIPU. This trade-off between the tenant VM CPU and CIPU highlights the need for flexible control, and we provide a per-VNI option to adapt to different deployment needs.

7.4 Pitfalls and Mitigation of Delayed Release

We face three major issues with sender-side delayed release.

Sender window configuration. Since delayed release retains packets in the VM's buffer until acknowledged, the window must not exceed the VM's buffer capacity, which is bounded by the default ring size of 4096 entries. In practice, we set the initial window to 512, as this value matches the bandwidth–delay product of the longest RTT in Alibaba Cloud, ensuring adequate depth without exceeding VM buffer limits.

Descriptor starvation under loss. Under heavy loss and retransmission, descriptors may remain unreturned for a long time. This quickly exhausts VM buffers and can trigger HoL blocking or even application-level failures. To address this, we configure a timeout: if a packet remains unacknowledged after three consecutive RTO failures (≈28 ms), the CIPU forcibly releases the descriptor and updates the *used ring*. This ensures timely buffer reclamation under persistent stalls, preventing buffer exhaustion and mitigating cascading failures.

Unsafe reuse of TX buffers. Although NAPI-TX [42] is enabled by default and widely used in Alibaba Cloud, we still need to handle corner cases where the guest VM OS disables NAPI-TX or employs user-space drivers (*e.g.,* DPDK [27]) that aggressively recycle buffers. In such cases, delayed release becomes unsafe: the CIPU may retransmit a buffer that has already been overwritten with new data, leading to corrupted packets. To solve this, we implement an adaptive fallback mechanism. Initially, delayed release is disabled and the CIPU caches full packets. Upon the first retransmission, the CIPU compares the current buffer checksum with the cached packet. If they match, indicating no early reuse as is typical under NAPI-TX, delayed release is enabled for subsequent transmissions. Otherwise, the connection remains in full-packet caching mode on CIPU to ensure correctness.

7.5 Applicable Workloads of BIFROST

In-memory databases (*e.g.,* Redis SET/GET). Such services are highly sensitive to tail latency and minor packet loss, and BIFROST's fast loss recovery significantly improves P999 response times, which is crucial for latency-critical applications such as financial trading and online gaming.

Global acceleration (GA) services. GA traffic traverses long-haul cross-region links where minute-scale losses and jitter frequently degrade performance and user experience. Deploying BIFROST between egress PoPs and the public cloud significantly improves reliability and stability.

Real-time video services. Real-time video streaming imposes strict frame deadlines, where transient packet loss or jitter directly manifests as buffering or quality degradation. BIFROST sustains smooth HD and 4K delivery by providing fast transport-layer recovery without application changes.

8 Related Work

Multipath transmission in data center. Multipath transmission has long been explored as a means to mitigate congestion and failures in datacenter networks. Some solutions focus on *flow-level* schemes such as PLB [54] and FlowBender [35], which dynamically schedule flows across multiple paths but suffer from coarse granularity and poor responsiveness under elephant flows. To capture finer dynamics, *flowlet-based* mechanisms including LetFlow [74] and HULA [37] split traffic into bursts (flowlets) and distribute them across paths, leveraging transient gaps in packet transmissions to reduce reordering. More recent designs move to *packet-level spraying*, such as SRD [59] , Strack [39], and Conweave [68], where packets are distributed across multiple available paths to provide fast failover and fine-grained balancing, and inevitably lead to more OOO packets. Similar ideas have also been explored in RDMA [21, 39, 65, 68], following the same design direction but with different implementation choices.

OOO transmission. Packet spraying is inherently OOO delivery, which requires reordering before packets can be consumed by the transport stack. Several works [2, 61, 62, 67, 68, 78] explore in-network packet reordering, leveraging programmable switches to align OOO packets. Some approaches [45, 65, 79] instead rely on reorder buffers at end-host NICs to handle OOO delivery, avoiding reliance on in-network devices. In contrast, some designs [12, 55] eliminate reordering by using Direct Data Placement (DDP), which places packets directly into target memory offsets.

Transport stack offload. Several production systems [39, 59, 65, 72] and research efforts have explored transport stack offload to improve reliability and latency. AWS SRD [59, 60] is a reliable transport protocol offloaded to AWS's custom Nitro cards [17], which uses packet-level spraying to provide low-latency communication in EFA and ENA Express [18, 19]. Google Falcon [65] is a hardware transport for Ethernet datacenters that supports multiple upper layer protocols with delay-based congestion control, multipath load balancing, and hardware retransmissions. Tesla TTPoE [72] is a hardware transport protocol deployed in the Tesla Dojo system [73], designed to replace TCP for large-scale machine learning workloads with low-latency transport over Ethernet. Ultra Ethernet Transport [33] is a hardware transport protocol by the Ultra Ethernet Consortium for scalable, low-latency, and reliable Ethernet communication in AI/ML and HPC networks.

9 Conclusion

In this paper, we present BIFROST, the next-generation VPC network with high-performance multipath RT. BIFROST addresses the long-standing challenges of building multipath RT in VPC networks: (i) mitigating network instabilities through RTT-aware multipath packet spraying, (ii) ensuring in-order delivery under limited SmartNIC memory via in-place guest reordering, (iii) providing end-to-end reliability and fast recovery through bitmap-based delayed ACK, and (iv) maintaining scalability with efficient reliability state management. Our extensive evaluation shows that BIFROST reduces tail latency by up to 307× for latency-sensitive services such as Redis. More-

over, our experience indicates that building high-performance multipath RT in VPC is not merely a protocol design problem but a systems-level problem, requires co-design across virtualization, hardware offload, and resource management. We believe the design principles distilled from BIFROST can guide both the academic community and other CSP in building high-performance multipath RT for large-scale clouds.

References

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 19, USA, 2010. USENIX Association.

[2] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. SP-PIFO: Approximating Push-In First-Out behaviors using Strict-Priority queues. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 59–76, Santa Clara, CA, February 2020. USENIX Association.

[3] Alibaba Cloud. Overview of Flowlog. https://www.alibabacloud.com/help/en/virtual-private-cloud/latest/flow-logs-overview, 2023.

[4] Alibaba Cloud. Traffic Mirroring Overview. https://www.alibabacloud.com/help/en/virtual-private-cloud/latest/traffic-mirroring-overview, 2023.

[5] Alibaba Cloud. Network ACL Overview. https://www.alibabacloud.com/help/en/vpc/network-acl-overview, 2024.

[6] Alibaba Cloud. VPC Route Table. https://www.alibabacloud.com/help/en/vpc/vpc-route-table/, 2024.

[7] Alibaba Cloud. Alibaba Cloud Cloud Enterprise Network (CEN). https://www.alibabacloud.com/en/product/cen, 2025.

[8] Alibaba Cloud. API Gateway. https://www.alibabacloud.com/en/product/api-gateway?_p_lc=1&spm=a3c0i.179194.6791778070.2.3e5a3a70e4Fnwy, 2025.

[9] Alibaba Cloud. Server Load Balancer. https://www.alibabacloud.com/en/product/server-load-balancer?_p_lc=1&spm=a3c0i.7911826.2564562790.4.6a323870SZ7vLu, 2025.

[10] Alibaba Cloud. Transit Router. https://www.alibabacloud.com/en/product/transit_router?_p_lc=1&spm=a3c0i.7911826.2564562790.1.6a323870SZ7vLu, 2025.

[11] Alibaba Cloud. Virtual Private Cloud (VPC). https://www.alibabacloud.com/en/product/vpc?_p_lc=1, 2025.

[12] Alibaba Cloud Documentation Center. Elastic Compute Service: eRDMA. Online; last updated August 1, 2025, 2025. Alibaba Cloud Help — User Guide, Network section: eRDMA.

[13] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: distributed congestion-aware load balancing for datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):503–514, August 2014.

[14] Amazon Web Services. Logging IP Traffic Using VPC Flow Logs. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html, 2021.

[15] Amazon Web Services. Amazon Virtual Private Cloud Traffic Mirroring. https://docs.aws.amazon.com/vpc/latest/mirroring/what-is-traffic-mirroring.html, 2022.

[16] Amazon Web Services. Amazon VPC Documentation. https://docs.aws.amazon.com/vpc/, 2024.

[17] Amazon Web Services. AWS Nitro System. https://aws.amazon.com/ec2/nitro/, 2025.

[18] Amazon Web Services. Elastic Fabric Adapter (EFA). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html, 2025.

[19] Amazon Web Services. ENA Express. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ena-express.html, 2025.

[20] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the seventh conference on emerging networking experiments and technologies*, pages 1–12, 2011.

[21] Tommaso Bonato, Abdul Kabbani, Ahmad Ghalayini, Michael Papamichael, Mohammad Dohadwala, Lukas Gianinazzi, Mikhail Khalilov, Elias Achermann, Daniele De Sensi, and Torsten Hoefler. Arcane: Adaptive routing with caching and aware network exploration, 2025.

[22] Olivier Bonaventure, Christoph Paasch, and Gregory Detal. Use cases and operational experience with multipath tcp. Technical report, 2017.

[23] Lucas Chaufournier, Ahmed Ali-Eldin, Prateek Sharma, Prashant Shenoy, and Don Towsley. Performance evaluation of multi-path tcp for data center and cloud workloads. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 13–24, 2019.

[24] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. The RACK-TLP Loss Detection Algorithm for TCP. RFC 8985, February 2021.

[25] Alibaba Cloud. A Detailed Explanation about Alibaba Cloud CIPU. https://www.alibabacloud.com/blog/599183, 2023.

[26] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637. IEEE, 2011.

[27] DPDK Project. Data Plane Development Kit (DPDK), 2024.

[28] Zihao Fan, Enge Song, Bo Jiang, Yang Song, Yuke Hong, Bowen Yang, Yilong Lv, Yinian Zhou, Junnan Cai, Chao Wang, Yi Wang, Yehao Feng, Dian Fan, Ye Yang, Shize Zhang, Xiaoqing Sun, Jianyuan Lu, Xing Li, Jun Liang, Biao Lyu, Zhigang Zong, and Shunmin Zhu. Understanding the long tail latency of tcp in large-scale cloud networks. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, APNET '25, page 106–113, New York, NY, USA, 2025. Association for Computing Machinery.

[29] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. Tcp extensions for multipath operation with multiple addresses. Technical report, 2013.

[30] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. Juggler: a practical reordering resilient network stack for datacenters. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16, 2016.

[31] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[32] Google Cloud. Virtual Private Cloud (VPC). https://cloud.google.com/vpc, 2025.

[33] Torsten Hoefler, Karen Schramm, Eric Spada, Keith Underwood, Cedell Alexander, Bob Alverson, Paul Bottorff, Adrian Caulfield, Mark Handley, Cathy Huang, Costin Raiciu, Abdul Kabbani, Eugene Opsasnick, Rong Pan, Adee Ran, and Rip Sohan. Ultra ethernet's design principles and architectural innovations, 2025.

[34] Intel Corporation. Infrastructure Processing Unit (IPU), 2024.

[35] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 149–160, 2014.

[36] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 323–335, 2017.

[37] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[38] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 514–528, 2020.

[39] Yanfang Le, Rong Pan, Peter Newman, Jeremias Blendin, Abdul Kabbani, Vipin Jain, Raghava Sivaramu, and Francis Matus. Strack: A reliable multipath transport for ai/ml clusters, 2024.

[40] Xing Li, Xiaochong Jiang, Ye Yang, Lilong Chen, Yi Wang, Chao Wang, Chao Xu, Yilong Lv, Bowen Yang, Taotao Wu, Haifeng Gao, Zikang Chen, Yisong Qiao, Hongwei Ding, Yijian Dong, Hang Yang, Jianming Song, Jianyuan Lu, Pengyu Zhang, Chengkun Wei, Zihui Zhang, Wenzhi Chen, Qinming He, and Shunmin Zhu. Triton: A flexible hardware offloading architecture for accelerating apsara vswitch in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 750–763, New York, NY, USA, 2024. Association for Computing Machinery.

[41] Xing Li, Enge Song, Bowen Yang, Tian Pan, Ye Yang, Qiang Fu, Yang Song, Yilong Lv, Zikang Chen, Jianyuan Lu, Shize Zhang, Xiaoqing Sun, Rong Wen, Xionglie Wei, Biao Lyu, Zhigang Zong, Qinming He, and Shunmin Zhu. Nezha: Smartnic-based virtual switch load sharing. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 218–233, New York, NY, USA, 2025. Association for Computing Machinery.

[42] Linux Kernel Community. NAPI — New API for network drivers. https://www.kernel.org/doc/Documentation/networking/napi.rst, 2025.

[43] Jingling Liu, Jiawei Huang, Wenjun Lv, and Jianxin Wang. Aps: Adaptive packet spraying to isolate mix-flows in data center network. *IEEE Transactions on Cloud Computing*, 10(2):1038–1051, 2020.

[44] Jianyuan Lu, Tian Pan, Shan He, Mao Miao, Guangzhe Zhou, Yining Qi, Shize Zhang, Enge Song, Xiaoqing Sun, Huaiyi Zhao, Biao Lyu, and Shunmin Zhu. Cloud-sentry: Two-stage heavy hitter detection for cloud-scale gateway overload protection. *IEEE Trans. Parallel Distrib. Syst.*, 35(4):616–633, April 2024.

[45] Jianyuan Lu, Shunmin Zhu, Jun Liang, Yuxiang Lin, Tian Pan, Yisong Qiao, Yang Song, Wenqiang Su, Yixin Xie, Yanqiang Li, Enge Song, Shize Zhang, Xiaoqing Sun, Rong Wen, Xionglie Wei, Biao Lyu, and Xing Li. Albatross: A containerized cloud gateway platform with fpga-accelerated packet-level load balancing. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 71–84, New York, NY, USA, 2025. Association for Computing Machinery.

[46] Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, and Dennis Cai. Alibaba stellar: A new generation rdma network for cloud ai. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 453–466, New York, NY, USA, 2025. Association for Computing Machinery.

[47] Microsoft Azure. Azure Virtual Network – Virtual Private Cloud. https://azure.microsoft.com/en-us/products/virtual-network, 2025.

[48] MongoDB Inc. MongoDB, 2024. [Online]. Available: https://www.mongodb.com/.

[49] NVIDIA. NVIDIA BlueField Networking Platform. https://www.nvidia.com/en-us/networking/products/data-processing-unit/, 2025.

[50] NVIDIA Corporation. DPA Subsystem — NVIDIA DOCA SDK. https://docs.nvidia.com/doca/sdk/DPA%2BSubsystem/index.html, 2025.

[51] Oracle Corporation. MySQL. https://www.mysql.com, 2024.

[52] 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. Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 194–206, New York, NY, USA, 2021. Association for Computing Machinery.

[53] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized" zero-queue" datacenter network. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 307–318, 2014.

[54] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. Plb: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 207–218, 2022.

[55] Mohammad J. Rashti and Ahmad Afsahi. 10-Gigabit iWARP Ethernet: Comparative Performance Analysis with InfiniBand and Myrinet-10G. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, 2007.

[56] Redis Ltd. Redis, 2024. [Online]. Available: https://redis.io/.

[57] Rusty Russell. virtio: towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, July 2008.

[58] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 151–162, 2013.

[59] Leah Shalev, Hani Ayoub, Nafea Bshara, Yuval Fatael, Ori Golan, Omer Ilany, Anna Levin, Zorik Machulsky, Kevin Milczewski, Marc Olson, et al. The tail at aws scale. *IEEE Micro*, 2024.

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

[61] Naveen Kr. Sharma, Ming Liu, Kishore Atreya, and Arvind Krishnamurthy. Approximating fair queueing on reconfigurable switches. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 1–16, Renton, WA, April 2018. USENIX Association.

[62] Naveen Kr. Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable calendar queues for high-speed packet scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 685–699, Santa Clara, CA, February 2020. USENIX Association.

[63] Dian Shen, Bin Yang, Junxue Zhang, Fang Dong, and John CS Lui. emptcp: A framework to fully extend multipath tcp. *IEEE/ACM Transactions on Networking*, 2024.

[64] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[65] Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan M. G. Wassel, Naveen Kr. Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, Sarin Thomas, Inho Cho, Hyojeong Lee Seibert, Baijun Wu, Rui Yang, Yuliang Li, Kai Huang, Qianwen Yin, Abhishek Agarwal, Srinivas Vaduvatha, Weihuang Wang, Masoud Moshref, Tao Ji, David Wetherall, and Amin Vahdat. Falcon: A reliable, low latency hardware transport. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 248–263, New York, NY, USA, 2025. Association for Computing Machinery.

[66] Shan Sinha, Srikanth Kandula, and Dina Katabi. Harnessing tcp's burstiness with flowlet switching. In *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*. Citeseer, 2004.

[67] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. Programmable packet scheduling at line rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 44–57, New York, NY, USA, 2016. Association for Computing Machinery.

[68] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. Network load balancing with in-network reordering support for rdma. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 816–831, New York, NY, USA, 2023. Association for Computing Machinery.

[69] Enge Song, Nianbing Yu, Tian Pan, Qiang Fu, Liang Xu, Xionglie Wei, Yisong Qiao, Jianyuan Lu, Yijian Dong, Mingxu Xie, Jun He, Jinkui Mao, Zhengjie Luo, Chenhao Jia, Jiao Zhang, Tao Huang, Biao Lyu, and Shunmin Zhu. Mimic: Smartnic-aided flow backpressure for cpu overloading protection in multi-tenant clouds. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, pages 1–11, 2022.

[70] Xiaoqing Sun, Xionglie Wei, Xing Li, Ju Zhang, Bowen Yang, Yi Wang, Ye Yang, Yu Qi, Le Yu, Chenhao Jia, Zhanlong Zhang, Xinyu Chen, Jianyuan Lu, Shize Zhang, Enge Song, Yang Song, Tian Pan, Rong Wen, Biao Lyu, Yang Xu, and Shunmin Zhu. Zooroute: Enhancing cloud-scale network reliability via overlay proactive rerouting. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 1251–1253, New York, NY, USA, 2025. Association for Computing Machinery.

[71] Igor Sysoev. Nginx: An HTTP and Reverse Proxy Server. https://nginx.org, 2004.

[72] Tesla. TTPoE: Tesla Transport Protocol over Ethernet. https://github.com/teslamotors/ttpoe, 2024. GitHub repository.

[73] Tesla, Inc. Tesla AI. https://www.tesla.com/en_gb/AI, 2025.

[74] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, 2017.

[75] Peng Wang, Hong Xu, Zhixiong Niu, Dongsu Han, and Yongqiang Xiong. Expeditus: Congestion-aware load balancing in clos data center networks. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 442–455, 2016.

[76] 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, and Wenzhi Chen. Achelous: Enabling programmability, elasticity, and reliability in hyperscale cloud networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 769–782, New York, NY, USA, 2023. Association for Computing Machinery.

[77] Xin Yin, Enge Song, Ye Yang, Yi Wang, Bowen Yang, Jianyuan Lu, Xing Li, Biao Lyu, Rong Wen, Shibo He, Yuanchao Shu, and Shunmin Zhu. vswitchlb: Stratified load balancing for vswitch efficiency in data centers. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, APNet '24, page 95–101, New York, NY, USA, 2024. Association for Computing Machinery.

[78] Zhuolong Yu, Chuheng Hu, Jingfeng Wu, Xiao Sun, Vladimir Braverman, Mosharaf Chowdhury, Zhenhua Liu, and Xin Jin. Programmable packet scheduling

with a single queue. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 179–193, New York, NY, USA, 2021. Association for Computing Machinery.

[79] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: reducing the flow completion time tail in datacenter networks. *SIGCOMM Comput. Commun. Rev.*, 42(4):139–150, August 2012.

[80] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2017.

[81] Tao Zhang, Ran Huang, Yuanzhen Hu, Yangfan Li, Shaojun Zou, Qianqiang Zhang, Xin Liu, and Chang Ruan. Load balancing with deadline-driven parallel data transmission in data center networks. *IEEE Internet of Things Journal*, 10(2):1171–1191, 2022.

[82] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, Qing An, Hai Hong, Hongqiang Harry Liu, and Ming Zhang. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 418–432, New York, NY, USA, 2021. Association for Computing Machinery.

Appendices

A VPC and VM Growth Trends

To provide context on the scale of our production environment, Figure A1 shows the growth of VPCs and VMs in Alibaba Cloud over the past five years. As the left panel illustrates, the number of VPCs has more than doubled from 3 trillion in 2020 to nearly 8 trillion in 2024. Meanwhile, the right panel shows that the number of VMs per VPC has also grown rapidly, reaching over 170 million by 2024. These trends underscore the role of VPC as the foundation of cloud networking, supporting massive tenant isolation and elastic resource management at scale.

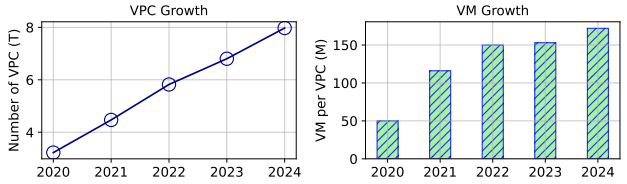

Figure A1: Growth of VPCs (left) and VMs per VPC (right) in Alibaba Cloud.

B Production Case Studies

In this section, we presents a series of representative operational incidents encountered during Alibaba Cloud's production operations, illustrating key failure scenarios, their root causes, and the resulting impact on customer services.

B.1 Host-Side Failure Cases

Bitter Case I: *Jun 2, 2022 — Monitoring detected a single-sided NIC fault on a customer server, causing intermittent timeouts in business container access to the database for approximately 176 minutes.*

Bitter Case II: *Nov 22, 2021 — Single-side NIC flapping on customer server caused intermittent network connectivity and business disruption, lasting approximately 141 minutes.*

B.2 Network-Side Failure Cases

Bitter Case III: *May 23, 2022 — Spine switch malfunction caused network packet loss and elevated error rates in business services, resulting in approximately 184 minutes of service disruption.*

Bitter Case IV: *Mar 4, 2022 — An integrated access switch anomaly caused intermittent packet drops, degrading latency-sensitive advertising services for over 60 minutes, despite the core-side fault lasting only ∼20 minutes.*

B.3 Middlebox Failure Cases

Bitter Case V: *Aug 16, 2024 — A sudden traffic surge overwhelmed the packet transformation of network load balancer, causing continuous packet losses for 14 minutes.*

C Protocol Specification

C.1 Data Packet Header

The BIFROST data packet header (Figure C2) encodes control and reliability information in a compact format. The header starts with the `Pkt Type` field (8 bits), indicating whether the packet is used for data transmission, acknowledgment, or connection management. It is followed by a `Version` field (4 bits) for protocol compatibility and a `CC Type` field (4 bits) specifying the active congestion control algorithm. A set of 1-bit control flags refine transport semantics: `se` marks special packet types such as keepalive, `ack` indicates whether acknowledgment is required, `dir` specifies the packet direction, `ece` enables explicit congestion notification, and `ret` identifies retransmitted packets. A 3-bit `Path ID` field identifies designated paths within a multipath connection and enables packet-level steering. An 8-bit `Reserved` field is preserved for future extensions. Beyond these control fields, the header carries a `Packet Sequence Number` (32 bits) to support reliable delivery, a `Destination CID` (32 bits) to identify the receiving connection, and a `Timestamp Value` (32 bits) for RTT estimation and congestion control.

Bits 0-7	8-11	12-15	16	17	18	19	20	21-23	24-31
Pkt Type	Version	CC Type	se	ack	dir	ece	ret	Path ID	Reserved
Packet Sequence Number (PSN)									
Destination Connection ID (CID)									
Timestamp Value									

Figure C2: BIFROST data packet header.

C.2 Connection Establishment Packet Header

The BIFROST connection establishment packet header (Figure C3) largely follows the same structure as the data packet header, beginning with the `Pkt Type` (8 bits), `Version` (4 bits), and `CC Type` (4 bits) fields, followed by a `Thread ID` (8 bits) to facilitate core affinity and an 8-bit Reserved field for future use. In addition to the `Destination CID` (32 bits) already carried by data packets, the header also includes a `Local CID` (32 bits). This extra identifier allows both endpoints to exchange and bind their CIDs during the handshake, ensuring consistent connection mapping for subsequent reliable transmission.

C.3 ACK Packet Header

The BIFROST ACK packet header consolidates both basic control fields and bitmap-based acknowledgment information. The header begins with `Pkt Type` (8 bits), indicating the packet type. The `Path ID` (3 bits) also identifies

Bits 0-7	8-11	12-15	16-23	24-31
Pkt Type	Version	CC Type	Thread ID	Reserved
Local CID				
Destination CID				

Figure C3: BIFROST connection establishment packet header.

designated paths of multipath transmission. The `Packet Sequence Number` (32 bits) and `Destination CID` (32 bits) carry the same semantics as in data packets. To support congestion control and RTT estimation, the header further embeds three timestamps: `Timestamp Value` (32 bits) generated at packet transmission, `Timestamp Echo` (32 bits) carrying the sender's timestamp for RTT calculation, and `Timestamp NIC RX` (32 bits) recording the NIC reception time. Additional lightweight fields include `ACK Type` (3 bits), `ACK Value` (5 bits) for extended acknowledgment semantics, `Hop Count` (4 bits) to measure the number of forwarding hops, and the `se` flag (1 bit) to mark special packet types such as keepalive.

On top of these, the header incorporates a bitmap-based acknowledgment scheme. Specifically, `Num_ACK` (11 bits) records the number of acknowledged packets, `Num_ECE` (11 bits) counts congestion marks, and `PSN_bitmap_len` (4 bits) specifies the bitmap length. The acknowledgment range is anchored by `PSN_base` (32 bits), followed by up to eight `PSN_bitmap[·]` (32 bits each) fields, which encode the reception state of consecutive packets. This compact structure enables a single ACK packet to cumulatively acknowledge up to 256 packets, providing fast and precise feedback for loss recovery and congestion control.

Bits 0-7	8-11	12-15	16	17-19	20-23	24-28	29-31
Pkt Type	Version	CC Type	se	Path ID	Hop Count	ACK Value	ACK Type
Packet Sequence Number (PSN)							
Destination CID							
Timestamp Value							
Timestamp Echo							
Timestamp NIC RX							
Num_ACK		Num_ece		PSN_bitmap_len		Reserved	
PSN_base							
PSN_bitmap[0]							
PSN_bitmap[1]							
PSN_bitmap[2]							
PSN_bitmap[3]							
PSN_bitmap[4]							
PSN_bitmap[5]							
PSN_bitmap[6]							
PSN_bitmap[7]							

Figure C4: BIFROST ACK header.

D Congestion Control

We adopt the delay-based congestion control (CC) algorithm of Swift [38], as delay is both an easily measurable signal in large-scale networks and a precise indicator of path quality.

***Timestamp* measurement.** To support Swift CC, BIFROST instruments fine-grained timestamping on both data and ACK

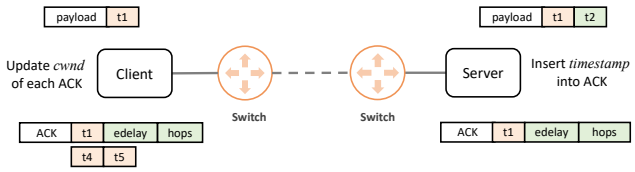

Figure C5: Timestamp-based delay measurement.

Table C1: Delay measurement for congestion control.

Symbol	*Description*
t_1	Client RTE transmits the data packet
t_2	Server AVS receives the data packet
t_3	Server RTE sends the ACK
t_4	Client AVS receives the ACK
t_5	Client RTE receives the ACK

packets (Figure C5). Each data packet carries a transmit timestamp t_1, while the receiver records t_2 when the AVS receives the packet and t_3 when the RTE generates the ACK. Upon ACK arrival at the client, the AVS records t_4 and the RTE records t_5. The definitions of these timestamps and their measurement points are summarized in Table C1. These timestamps allow decomposing the end-to-end RTT into different components: the total RTT is $RTT = t_5 - t_1$; the endpoint delay is computed as $edelay + (t_5 - t_4)$, where $edelay = t_3 - t_2$ captures server-side processing; and the *fabric-delay* is derived as RTT minus the endpoint delay. These timestamps allow decomposing the end-to-end RTT into different components. First, the total round-trip time is:

$$RTT = t_5 - t_1$$

Next, we isolate the server-side processing delay as:

$$edelay = t_3 - t_2$$

Using this, the endpoint delay, which includes both server processing and client receive delay, can be expressed as:

$$\textit{Endpoint-delay} = edelay + (t_5 - t_4)$$

Finally, the remaining portion of RTT corresponds to the in-fabric latency:

$$\textit{Fabric-delay} = RTT - \text{Endpoint delay}$$

In addition, the number of traversed hops is obtained at the AVS by reading the IP header's TTL field. BIFROST feeds these measurements into the Swift CC algorithm [38] to update the congestion window (*cwnd*) of each path.

Integration with BIFROST RTE. As illustrated in Figure **??**, CC module is tightly integrated into the BIFROST RTE. When a new flow arrives, the scheduler queries the CC module to retrieve the corresponding *cwnds* of the selected paths, which guides packet admission and scheduling. Upon receiving ACKs, the RTE extracts key fields including the *CID*, *path ID*, and *timestamps*, and forwards them to the CC module.

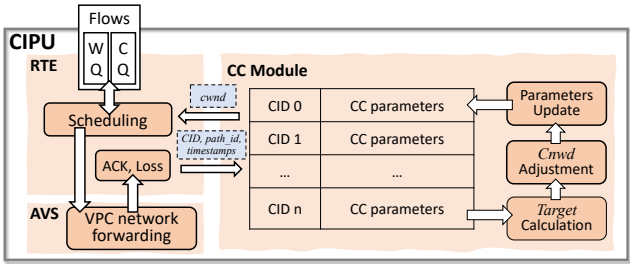

Figure D6: Integration of CC into BIFROST RTE.

The CC module locates the corresponding per-connection state, computes the *target-delay*, adjusts the *cwnd* of the specified path, and updates CC parameters accordingly [38]. This design offers three advantages: it minimizes the exposed interface to the RTE, enables clean decoupling such that the RTE can be readily offloaded to NIC hardware while the CC module remains in software, and provides extensibility by allowing different CC algorithms to be supported through the same interface beyond Swift.

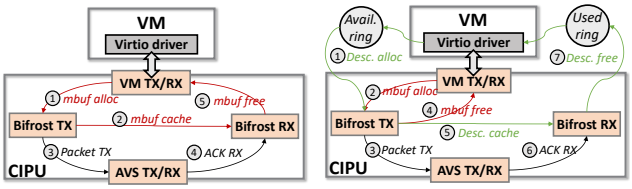

Figure D7: Sender-side delayed release mechanism: (Left) without delayed release and (Right) with delayed release.

E Detailed Pipeline of Delayed Release

The most straightforward approach to achieving reliable packet transmission is to let the CIPU cache all unacknowledged packets in its local memory, as illustrated in Figure D7 (left). Specifically, when the CIPU sends a packet, it retains the full payload in memory (②) until the ACK is received from the receiver, after which the buffer is finally released (⑤). Under high connection concurrency, this quickly consumes memory and is particularly challenging given the CIPU's limited capacity. To address this, BIFROST introduces a sender-side delayed release mechanism that shifts the packet buffering responsibility from the CIPU to the tenant VM, as shown in Figure D7 (right). When transmitting a packet, the CIPU first fetches the *descriptors* from the avail ring (①). Instead of caching the full packet in its local memory, the CIPU only stores the lightweight descriptors for potential retransmission (④⑤), while the actual packet remains pinned in the VM's memory. Once an ACK is received (⑥), the CIPU writes the *descriptors* into the *used ring* (⑦), allowing the VM to reclaim the buffer. Since *descriptors* are much smaller than full packets, this design greatly reduces CIPU memory consumption and improves scalability. This mechanism incurs no additional memory overhead on the VM, as its protocol stack (*e.g.,* TCP) already retains the full packet until the corresponding ACK arrives. BIFROST leverages this fact to eliminate redundant packet buffering on the CIPU.

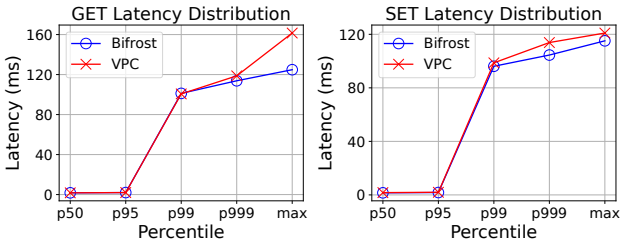

Figure E8: Redis short connection performance.

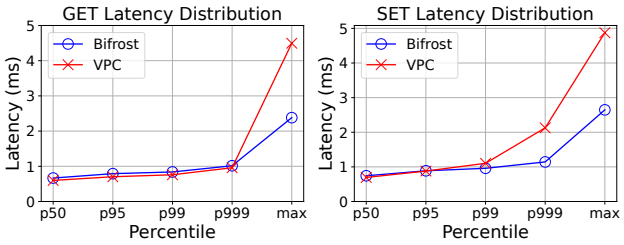

Figure E9: Redis persistent connection performance.

F Supplementary Experiments

Redis under stable conditions. We further evaluate Redis in the short-connection mode without injecting artificial loss, using the same setup of two VMs placed on different physical servers. Figure E8 reports the latency distribution for both *get* and *set* operations. Compared to the vanilla VPC, BIFROST incurs negligible overhead in average latency and reduces tail latency at high percentiles (P999 and Max), showing that its mechanisms remain lightweight under relatively stable network conditions. Figure E9 also reports the latency distribution for both *get* and *set* operations of persistent connections, which exhibit the same trend: BIFROST incurs only minimal overhead at lower percentiles while further reducing tail latency at high percentiles.

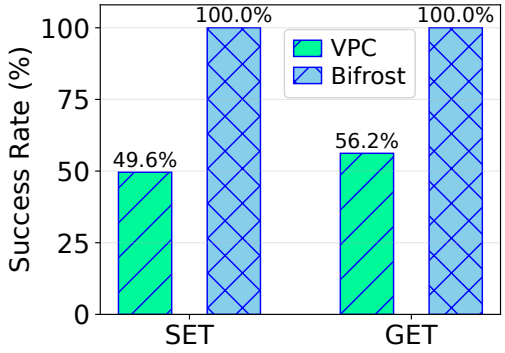

Figure F10: Redis request success rate.

Redis request success rate. We further evaluate request success rate when inducing a physical NIC port failure. The experiment uses two VMs on different physical servers, where

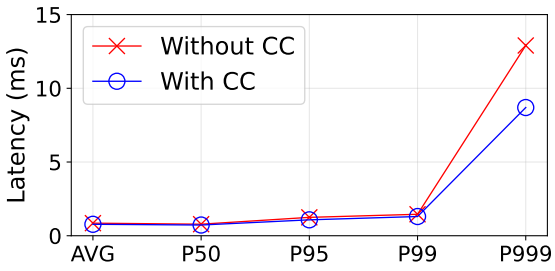

Figure F11: Impact of CC on Redis latency.

the client issues *get* and *set* requests to the server to test request success rate. As shown in Figure F10, under the vanilla VPC the success rate drops to 49.6% for *set* and 56.2% for *get*, as requests are disrupted by the failure. In contrast, BIFROST masks the port failure by rerouting traffic across healthy paths, sustaining a 100% success rate for both operations.

Redis performance under congestion control. We evaluate Redis with 200 concurrent clients issuing short-message persistent *Set* requests. To emulate network instability, the receiver-side AVS is configured with a 0.1% packet drop rate. We compare BIFROST with CC enabled and disabled, reporting latency across different percentiles (Figure F11). With CC enabled, the average to P99 latencies exhibit modest reductions, while the P999 latency shows a more pronounced 30% improvement.