Understanding the Long Tail Latency of TCP in Large-Scale Cloud Networks

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[‡], Shunmin Zhu[*‡¶]

[†]Shanghai Jiao Tong University [‡]Alibaba Cloud [§]Zhejiang University [*]Hangzhou Feitian Cloud

{fzhsjtu2023,bjiang}@sjtu.edu.cn,alibaba_cloud_network@alibaba-inc.com

Abstract

In today's cloud networks, TCP dominates traffic due to its reliability. However, TCP's inherent inefficiencies fail to effectively handle the inevitable instabilities of large-scale cloud networks, resulting in long tail latency. As a major cloud vendor, we've received numerous customer reports of long tail latency significantly degrading application performance. Despite extensive efforts, existing solutions face various deployment issues that limit their adoption in cloud networks. To this end, we propose Bifrost, a reliable transport protocol specifically optimized for TCP performance in cloud networks. Bifrost leverages techniques including RTT-aware multipath transmission, hybrid hardware-software packet reordering, optimized ACK mechanisms, and rapid loss detection. Our comprehensive evaluations demonstrate that Bifrost achieves an average 18.8x reduction in P99 latency compared to native TCP in various packet loss scenarios.

CCS Concepts

• **Networks → Data center networks**; **Network performance analysis**; **Reliable Transport Protocols**.

Keywords

TCP, Tail Latency, Cloud Networks

ACM Reference Format:
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, Shunmin Zhu. 2025. Understanding the Long Tail Latency of TCP in Large-Scale Cloud Networks. In *9th Asia-Pacific Workshop on Networking (APNET 2025), August 07–08, 2025, Shang Hai, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3735358.3735393

1 INTRODUCTION

Cloud vendors handle millions of workloads every day. As a major cloud vendor, our statistics show that 84.7% of workloads require reliable data transmission, typically achieved using the TCP protocol. We further find that tail latency-sensitive TCP services make up over 50% of all our services and 60% of our TCP services, including applications like Redis [25], Nginx [32], and MySQL [21]. However, we have received thousands of customer reports showing that long tail latency—the slowest request that significantly exceeds average response times—directly degrades the performance, causing service disruptions, SLA violations, and millions in revenue losses for both cloud vendors and customers.

Long-tail latency in cloud networks is caused by a mismatch between the instability of large-scale infrastructure and the limitations of TCP. To better understand this issue, we conduct an in-depth analysis of the root causes behind long tail latency in cloud networks. The large scale of cloud infrastructure, with its millions of links and tens of thousands of network devices in one region, introduces unavoidable instabilities such as packet loss and jitter. In a typical region with approximately *100,000* servers, our production data clearly demonstrate this phenomenon: network jitter occurs *hundreds* of times daily, elephant flows appear *thousands* of times weekly, physical NIC flapping happens *millions* of times daily, and frequent packet loss manifests at various points throughout the network. However, TCP's inherent inefficiencies, including single-path transmission, slow packet loss detection and delayed congestion control, fail to effectively handle these instabilities, directly resulting in the long tail latency issues observed in our cloud networks. Our tests reveal that a 1% packet loss rate, a common occurrence in cloud networks, TCP's P99 latency increases by over 150× compared to lossless scenarios.

Despite numerous academic and industry efforts to address long tail latency issues [1, 2, 4, 7, 9–15, 17, 18, 20, 24, 26–30, 33–40], existing solutions still face various deployment challenges that limit their practical adoption in cloud networks. Some approaches offer *limited performance improvements*, like PLB [24] and LetFlow [35] that use random path selection during repath, and MPTCP [10, 29] that suffers from Head-of-Line (HOL) blocking issues. Some approaches introduce *intrusiveness to users*, such as Clove, PLB, and Flowbender [13, 14, 24] require users to enable explicit congestion notification (ECN), and implementations such as MPTCP, PLB, and Flowbender [10, 13, 24] require kernel modifications. Dynamic routing solutions [2, 13, 15, 35] have *poor compatibility* because they require specialized switch support, which is difficult to guarantee in large-scale networks due to the heterogeneity of network devices. Some other solutions are *incompatible with cloud networks*, as they are tailored for single-tenant AI training scenarios [7, 28, 33], where hardware is homogeneous and traffic patterns are predictable, assumptions that do not hold in multi-tenant large-scale cloud networks.

Based on our analysis and the limitations of existing approaches, we identify that the core challenge is *mitigating long-tail latency*

[¶]Co-corresponding authors.

in unstable, large-scale cloud networks while maintaining complete transparency for end users. To address this, we propose Bifrost, a transport protocol that introduces a reliable transport layer between the application and VPC layers. Bifrost uses *RTT-aware multipath transmission* to handle elephant flows and transient link issues like jitter and flapping. To combat the packet out-of-order caused by multipath, we introduce *hybrid software-hardware packet ordering*. For enhanced reliability, we design *optimized ACK mechanisms* and enable *microsecond-precision packet loss detection* to detect and recover from loss across the entire data path. Extensive evaluations demonstrate that Bifrost significantly outperforms native TCP, maintaining low tail latency even under adverse network conditions. We summarize our contributions as follows:

- We analyze the root causes of TCP's long tail latency in large-scale cloud networks. Through production data, we demonstrate the severity of unavoidable instabilities in cloud networks and how TCP fails to effectively handle these instabilities, causing long tail latency.
- We propose Bifrost, a transport protocol that reduces long tail latency while maintaining user transparency and scalability, provides capabilities like multipath transmission and rapid packet loss detection.
- Our evaluations demonstrate that Bifrost achieves an average 18.8x reduction in P99 latency compared to native TCP in various packet loss scenarios.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the limitations of TCP (§2.1) and the network conditions observed of our cloud (§2.2 and §2.3). We then discuss current solutions and their limitations (§2.4). Finally, we present our design goals for addressing these challenges (§2.5).

2.1 High Tail Latency of TCP Flows

We have identified several critical challenges with TCP, which become increasingly prominent as cloud applications demand lower latency and higher throughput. These issues have been extensively studied in both academic research and industry deployments.

Low Bandwidth Utilization. TCP's in-order delivery requirement forces flows to use a single path when multiple paths are available. This is particularly problematic for elephant flows, which can saturate the capacity of a single path.

Slow Loss Detection and Recovery. TCP's loss recovery relies on receiving triple duplicate ACKs or triggering retransmission timeouts (RTO). The former requires multiple subsequent packets to arrive after a loss, while default RTO values (typically 200ms) are often much larger than typical RTTs.

Delayed Congestion Control. TCP detects congestion through packet loss, but by then networks are typically already severely congested, significantly impacting latency and preventing proactive congestion management. Moreover, packet loss can occur due to factors other than congestion, such as hardware failures or transmission errors at network devices.

2.2 Latency-Sensitive Service Distribution

We present the distribution of services across all applications running on the ECS (Elastic Compute Service) infrastructure in a region

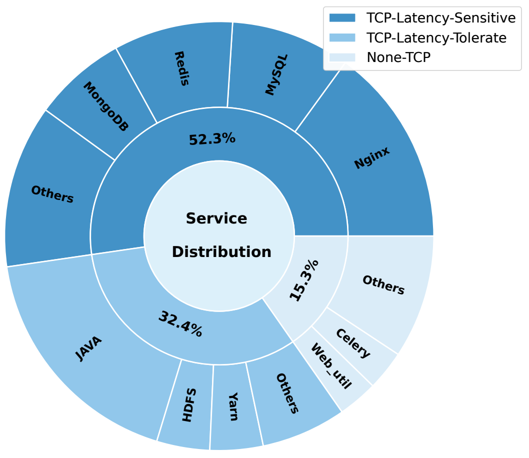


Figure 1: Service distribution in a region of Alibaba Cloud.

Table 1: Weekly CPU overload occurrence.

	Week1	*Week2*	*Week3*	*Week4*
Overload Counts	4701	6357	6051	5981

of Alibaba Cloud. As shown in Figure 1, TCP-based applications dominate the service landscape, accounting for 84.72% of all services. Among these TCP services, more than 60% are tail latency sensitive, including critical applications such as Redis, Nginx, MySQL, and MongoDB. These applications typically also have high bandwidth requirements, which have previously been constrained by TCP stack limitations and link bandwidth capacity. For example, a single slow operation in Redis can block subsequent requests, causing cascading delays that affect the entire service chain, making it particularly sensitive to tail latency issues. Additionally, Redis often handles large data transfers for caching and data synchronization, which also requires high bandwidth to maintain optimal performance and avoid data transfer bottlenecks. Given that TCP-based tail latency sensitive services constitute a significant portion of workloads in the cloud networks, addressing the inherent limitations of TCP protocol is crucial for providing low-latency, high-bandwidth data transmission capabilities for these services.

2.3 Cause Analysis of High Tail Latency

We conduct an in-depth analysis of the causes behind long tail latency in cloud networks. Due to the scale of cloud networks with numerous network devices and links, instabilities such as jitter and packet loss are inevitable. However, TCP's inability to efficiently handle these instabilities directly contributes to long tail latency. We will illustrate it through monitoring data and experiments.

2.3.1 Network Instability. **Elephant Flows.** Elephant flows are large, long-lived network flows that contribute disproportionately to the total traffic volume, often accounting for a small number of flows but a large percentage of bytes transferred. When elephant flows appear, TCP single-path transmission limitations force all

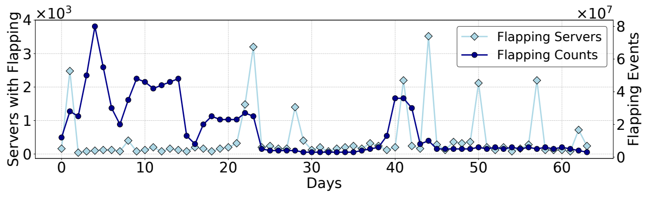


Figure 2: Physical NIC flapping over two months.

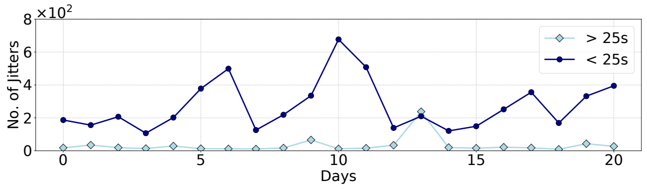


Figure 3: Physical network jitter over two months.

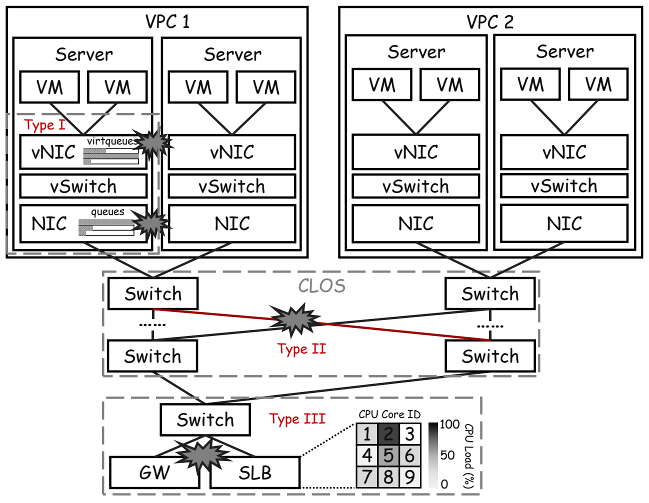


Figure 4: Root cause analysis of packet loss.

traffic through the same path and CPU core. Their massive volume and long duration overload specific CPU cores, creating processing bottlenecks that lead to packet drops. We conduct a month-long monitoring of all vSwitches in one region[1], and the results shown in Table 1 indicate that there are *thousands* of occurrences per week where CPU utilization exceeded 80%, with a significant proportion of these events caused by elephant flows according to [19, 22, 31].

Physical NIC Flapping. Physical NIC flapping manifests as intermittent connection failures in our cloud environment. These intermittent interface state changes disrupt ongoing network connections, as we observed applications being forced to re-establish their communication paths. Based on our deployment experience, these lead to additional latency and network instability, impacting application performance and user experience. As shown in Figure 2, we collect physical NIC flapping data in our cloud[2] over a two-month period. Our monitoring data reveal daily peak NIC flapping affecting *thousands* of servers, with *millions* of occurrences per day across our cloud infrastructure.

Physical Network Jitter. In large-scale networks, jitter refers to the variation in packet arrival times due to dynamic factors such as transient congestion, dynamic queue occupancy, scheduling variances, and hardware-level processing delays. From our years of deployment experience, these variations lead to irregular data transmission patterns that directly impact service quality and user experience. We collect physical network jitter data from cloud networks over a two-month period in one region[1], as shown in Figure 3. Our observations reveal *hundreds* of jitter events less than 25 seconds that occur daily, while events exceeding 25 seconds occurred *dozens* of times per day. The frequency and duration of these jitter events indicate that network instability is a persistent issue in the cloud.

Packet Loss. Packet loss occurs at various points throughout networks. For instance, in VM-to-VM traffic across different VPCs, packets must traverse a complex path of VM-vSwitch-Gateway-vSwitch-VM, creating multiple opportunities for packet loss at each transition point. As Figure 4 shows, our years of deployment experience reveal three main types of packet loss occurring at different points in the network path. We conduct comprehensive monitoring of packet loss in our cloud and present our measurement results.

Intra-Server Packet Loss (Type I). Packet loss occurs within the physical server, primarily at the physical NICs and virtual NICs (vNIC). *Packet loss in physical NICs* occurs when a large flow causes the buffer of the assigned receive queue to overflow. Although modern NICs hash packets across queues, a high-bandwidth flow is hashed to a single queue, leading to buffer exhaustion and packet drops, even if other queues remain underutilized. Since each queue is typically assigned to a dedicated CPU core, excessive traffic can also overload the core, further slowing packet processing and accelerating buffer overflow. As illustrated in Figure 5, we collect packet loss data from all physical NICs deployed across our cloud infrastructure[2] over a 20-day period, and our monitoring results highlight the severity of packet loss. For each server, we compute per-minute packet loss, and define a loss event as any minute during which the loss rate exceeds 1%. Under this definition, *tens of thousands of* servers experience loss events daily, amounting to *millions* of such events cloud-wide per day. At a lower threshold of 0.1%, the impact becomes even more widespread, affecting *hundreds of thousands of* servers daily and resulting in *tens of millions* of loss events across the cloud each day. Beyond packet loss in physical NICs, virtualized environments face another critical challenge: *Packet loss in vNICs.* In virtual machines, vNIC traffic processing relies on the host CPU. When the vCPU resources are insufficient or processing is too slow under high traffic rates, the virtual NIC's buffer overflows, resulting in packet loss. Our monitoring data from a region[1] reveal concern for packet loss across different virtualization solutions. The state-of-the-art SmartNIC solution (SmartNIC II) deployed in our cloud exhibits notably high packet loss rates, with the P99 and P999 across all servers in this region reaching 0.29% and 2%, respectively, over a 12-hour period, as shown in Table 2.

Packet Loss in Physical Networks (Type II). Due to the typically high bandwidth of physical links and network devices, packet loss

[1]This region contains approximately 100,000 servers.
[2]The entire cloud contains over one million servers.

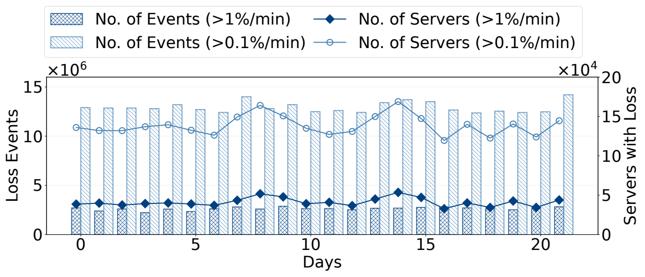


Figure 5: Packet loss in physical NICs over 20 days.

Table 2: Regional packet loss in vNICs over 12 hours.

Solution	*Average*	*P90*	*P99*	*P999*
Kernel-based VM	0.015%	0%	0.17%	2.0%
SmartNIC I	0.006%	0%	0.02%	1.0%
SmartNIC II	0.072%	0.03%	0.29%	2.0%

in physical networks is generally not caused by insufficient bandwidth, but rather by hardware issues such as physical link failures or network equipment malfunctions. Based on our years of deployment experience, the frequency of Type II packet loss is significantly lower than that of Type I and Type III.

Packet Loss in Middlebox (Type III). In the packet transmission path, traffic traverses various middleboxes such as gateways (GW) and server load balancers (SLB). These devices, typically running on ECS, have limited processing capabilities. Type III packet loss occurs when CPU cores assigned to specific flows become saturated, leading to processing bottlenecks, buffer overflows, and ultimately packet drops. We conduct a month-long monitoring of unplanned packet loss in CGW (Cloud Gateway), which primarily handles cross-region VPC traffic and ingress/egress traffic between the cloud and external networks, as shown in Figure 6. Although the loss probability remains below 0.01%, the massive traffic volume handled by Alibaba Cloud results in a sustained rate of *tens of thousands* of packets per second (PPS) being dropped. Despite the relatively low loss rate, such packet losses can still significantly degrade service quality and user experience.

2.3.2 *How Instability Degrades TCP Latency and Throughput.*

Network instability such as elephant flows, NIC flapping, and jitter ultimately manifest as packet loss that TCP must handle. The inherent detection of packet loss of TCP, which relies on duplicate ACK and large RTO values, significantly degrades performance. To evaluate the impact of packet loss on TCP-based applications, we deploy two virtual machines within the same VPC. Nginx short connection tests evaluate request latency while Sockperf measures throughput under various packet loss rates. Our Nginx short connection tests, as illustrated in Figure 7a, reveal that packet loss significantly increases request latency. Even with a minimal packet loss rate of 0.5%, the average latency increases sharply from 1 ms to around 10 ms. At 1% packet loss, the average latency reaches 12 ms and continues to deteriorate as packet loss increases, reaching 17.5 ms at 5% packet loss. The P99 latency exhibits an even more dramatic deterioration - with just 0.5% packet loss, it spikes to approximately 175 ms and plateaus at around 200 ms as the loss rate increases

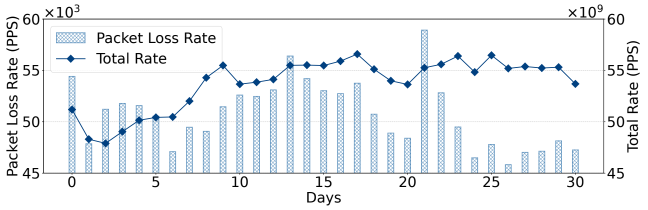


Figure 6: Packet loss in GW within a month.

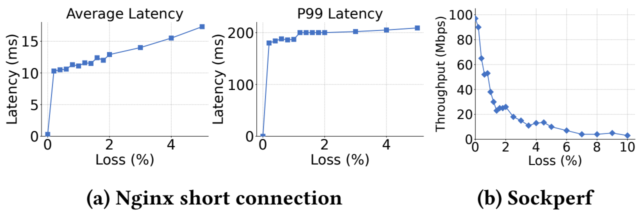


Figure 7: TCP-based application performance.

further. This substantial increase in tail latency is particularly problematic for tail latency sensitive applications that require consistent performance. The impact on throughput is equally concerning. As shown in Figure 7b, our test results demonstrate the severe impact of packet loss on Sockperf throughput: As the network packet loss rate increases from 0% to 1%, Sockperf throughput falls from 100Mbps to approximately 30Mbps. The degradation continues as packet loss increases, with throughput dropping below 10Mbps when packet loss exceeds 8%. Of greater concern, our monitoring data in § 2.3.1 indicates that 1% packet loss is a frequent occurrence in cloud networks. This indicates the severe performance degradation, both in terms of latency and throughput, is not merely a theoretical edge case but a common operational challenge.

2.4 Potential Solutions and Their Issues

While numerous efforts have been made to address the above challenges [1, 2, 4, 7, 9–15, 17, 18, 20, 24, 26–30, 33–40], existing solutions still face various deployment issues that limit their practical adoption in cloud networks. We categorize these issues in existing work into four main types:

Limited Performance Improvement. Solutions like PLB [24] and Letflow [35] use random path selection during repath, resulting in slower convergence and potentially selecting even worse paths. MPTCP [10, 29] suffers from HOL blocking issues during reordering of the end host packet and does not demonstrate clear benefits in cloud settings [5]. SRD [27, 28] adopts packet spraying but lacks reordering support at the receiver, leading to unnecessary retransmissions due to out-of-order packets.

Intrusiveness to Users. Solutions like Clove, PLB and Flowbender [13, 14, 24] rely on ECN, which is typically disabled by default on end hosts [31], requiring user intervention. MPTCP, PLB needs modifications to the user's kernel, making it intrusive to deploy.

Poor Compatibility and Scalability. Dynamic routing [2, 13, 15, 35, 36] lacks compatibility as it depends on customized switch functionalities, which cannot be supported across all switches in large-scale cloud networks. Centralized approaches [1, 3, 8, 23] depend on central servers to process control messages, likewise lacking scalability for large-scale deployments.

Incompatibility with Cloud Networks. Many existing works, such as Falcon, TTPoE, and SRD [7, 28, 33], primarily target AI training workloads. These environments typically feature single-tenant deployments, predictable traffic patterns, abundant bandwidth, and negligible packet loss. In contrast, cloud networks predominantly carry socket-based traffic, characterized by multi-tenancy and highly unpredictable workloads. In cloud networks, where tenants share physical bandwidth, redundant retransmissions, such as those in Go-Back-N, incur high overhead, making fine-grained loss recovery essential. As a result, solutions designed for AI training are incompatible with the requirements of cloud network.

2.5 Challenges & Design Goals

Through the above analysis, we find that the main challenge is *mitigating long tail latency in unstable, large-scale cloud networks while maintaining complete transparency to end users.* To address it, we establish the following design goals:

High Performance. Providing low-latency, high-bandwidth transmission capabilities for TCP-based latency-sensitive applications, which make up more than half of cloud networks.

Scalable for All Devices & Transparent to Users. Designing for large-scale multi-tenant cloud networks that operate seamlessly over heterogeneous network devices, while maintaining complete transparency to applications and end users.

Robust Against Network Instability. Efficiently handle instability in cloud networks, such as link failures and various packet loss, by providing rapid failover and recovery mechanisms.

3 SYSTEM DESIGN

In this section, we introduce the design of BIFROST, as shown in Figure 8. We first show how BIFROST addresses our design goals, then detail its components.

3.1 Design Overview

BIFROST directly addresses each key challenge with specifically tailored mechanisms, as summarized below:

- To achieve high performance under diverse network conditions, we design an RTT-aware multipath transmission mechanism and implement precise RTO control to minimize retransmission delays.
- To ensure scalability and user transparency, BIFROST inserts a transport layer between the application layer and the VPC layer at end hosts, which requires no changes to the underlying network devices and remains non-intrusive to user.
- To handle inherent network instability, BIFROST employs delayed ACK with bitmap aggregation and bitmap-based loss detection, enabling rapid detection of packet loss.

3.2 BIFROST Architecture

3.2.1 RTT-Aware Multipath Transmission. BIFROST implements RTT-aware packet-level multipath transmission at the end host. BIFROST performs RTT probing for each available path based on the traffic packets transmitted over those paths, steering traffic away from failed or degraded links. Each flow is divided into packet groups based on a predefined granularity and transmitted across

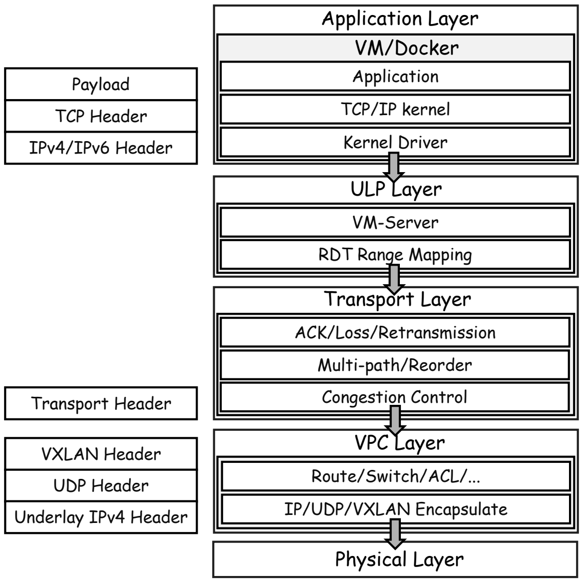
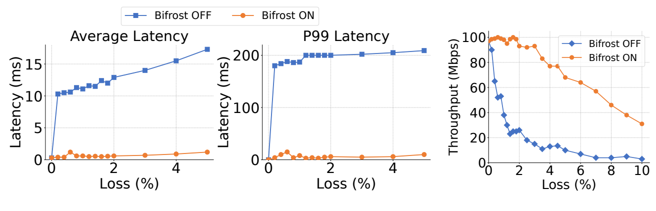


Figure 8: BIFROST protocol stack.

available paths, with the algorithm prioritizing paths with the lowest RTT. To minimize repath overhead, we employ a path stickiness optimization: when transmitting consecutive packet groups, BIFROST compares the RTT of the current path with the best RTT identified so far. If the RTT difference (δ) is below a defined threshold (e.g., $20\mu s$), the current path is maintained; otherwise, BIFROST switches to the path with the lowest RTT. When receiving an ACK, BIFROST extracts the measured RTT and the corresponding path from the ACK. Additionally, if a path has not transmitted packets for a prolonged period, BIFROST sends a probing packet to measure the RTT, ensuring path liveness and up-to-date RTT information. This approach effectively balances the trade-off between repath overhead and transmission efficiency while maintaining real-time adaptability to network conditions.

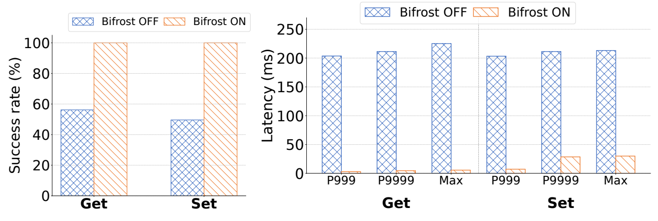
3.2.2 Hybrid Hardware-Software Reordering. To address packet reordering caused by multipath transmission, we reorder packets before delivering them to the guest. Traditional methods rely on the protocol stack to reorder full packets, leading to high resource usage and poor performance. We design a *hybrid hardware-software reordering mechanism*, leveraging CIPU's [6] architecture to perform efficient reordering at the receiver. BIFROST leverages the hardware buffer in CIPU to handle incoming packets, while the SoC performs packet sequence number (PSN)-based sorting to ensure in-order delivery. To improve data processing efficiency and minimize SoC memory pressure, only packet metadata—rather than full packets—is retrieved from the hardware buffer according to the sorted PSN sequence and delivered to the protocol stack. After the protocol stack processes the metadata, the corresponding packets are fetched from the hardware buffer and delivered to the guest.

3.2.3 ACK Aggregation via Delayed Bitmap. To enable end-to-end loss detection across the entire packet transmission path, we design an *ACK Aggregation via Delayed Bitmap* mechanism. Specifically,



(a) Nginx short connection (b) Sockperf

Figure 9: TCP-based app performance optimization.



(a) Success rate (b) Request latency

Figure 10: Redis performance optimization.

ACKs are generated only after packets have been successfully received by the vNIC, rather than immediately upon arrival at the physical NIC. This ensures that packet loss within the virtual NIC layer is also captured. The aggregate ACK mechanism employs a bitmap to efficiently represent the reception status of multiple packets in a single ACK, enabling the sender to quickly identify potential packet losses through the bitmap pattern.

3.2.4 *Bitmap Loss Detection & Precise RTO.*

To accurately identify lost packets that require retransmission, we design a *rapid packet loss detection mechanism* based on PSN bitmap analysis and microsecond-precision timeout retransmission. Upon receiving an ACK bitmap, Bifrost analyzes the bitmap to detect discontinuities in the PSN sequence. If gaps are identified in the PSN sequence, Bifrost triggers fast retransmission immediately. Furthermore, for tail losses, where packets are lost at the end of a transmission window, Bifrost employs microsecond-granularity timeout detection. When the delay of an unacknowledged packet exceeds 4 milliseconds (about 2 RTT), Bifrost initiates retransmission, far more precise than TCP's RTO of hundreds of milliseconds. The precise RTO and immediate loss detection enable rapid packet recovery and significantly reduce tail latency.

3.2.5 *Latency-Based Congestion Control.*

While native TCP relies on packet loss for congestion detection, which introduces significant latency in control responses, our ongoing work explores a latency-based approach. Since latency is the most easily measurable signal in cloud networks, and its variation can predict congestion events ahead of time, we can proactively adjust the transmission behavior based on these predictions. Similar to Swift [16], we are developing a mechanism that uses latency as a signal to detect congestion and adjust window sizes. Our approach monitors both fabric and endpoint latency separately to identify congestion bottlenecks with greater precision. Bifrost aims to enable more responsive congestion control than conventional loss-based methods.

4 PRELIMINARY EVALUATION

Experiment Setting. We evaluate Bifrost on a testbed consisting of two KVM-based VMs with 8 vCPUs and 32GB RAM deployed within the same VPC of Alibaba Cloud.

Performance Improvements. To assess the performance of Bifrost for TCP-based applications, we evaluate latency using Nginx short connections and throughput using Sockperf. As Figure 9a shows, with 1% packet loss, Bifrost achieves an average latency of 1.2 ms compared to 12 ms without it. The improvement is even greater for tail latency, with P99 measurements showing less than 10 ms with Bifrost versus 200 ms without it. Sockperf results further highlight the effectiveness of Bifrost. Figure 9b shows that Bifrost maintains robust performance under adverse network conditions, achieving 97 Mbps throughput at 1% packet loss—more than three times the 30 Mbps observed without it. This resilience extends to higher packet loss rates, with Bifrost delivering 65 Mbps throughput even at 5% loss.

Redis Request Performance Evaluation. To further validate the performance of Bifrost, we conduct Redis request tests to evaluate its effectiveness in handling critical applications in cloud networks. As shown in Figure 10a, during network failures, Bifrost maintains a nearly 100% success rate for both Get and Set requests, compared to approximately 50% without it. As shown in Figure 10b, Bifrost maintains low tail latency under congestion, achieving a P999 latency of approximately 3ms for Get requests and 7ms for Set requests, representing a significant improvement over the latency of more than 200ms observed without Bifrost.

5 OPEN ISSUES

(1) Swift-like congestion control mechanisms exhibit performance limitations in dynamic cloud networks, especially in their slow recovery of congestion windows following window reduction. (2) Currently, Bifrost only supports TCP socket semantics, which limits its applicability. Future development should expand protocol support beyond TCP to include RDMA, NVMe, and other application protocols, enhancing its versatility in cloud networks.

6 CONCLUSION

Severe TCP tail latency issues, caused by inevitable network instabilities and TCP's inherent inefficiencies, continue to degrade application performance in large-scale cloud networks. Bifrost offers a comprehensive solution to the long tail latency challenges of TCP-based applications in cloud networks. Through its RTT-aware multipath transmission, optimized ACK mechanisms, and comprehensive and rapid loss detection, Bifrost significantly improves performance, maintaining low tail latency in adverse network conditions. Our evaluations confirm its superiority over native TCP, demonstrating its potential to address critical challenges in large-scale multi-tenant cloud networks.

Acknowledgments

We are sincerely grateful to all anonymous reviewers for their valuable and constructive comments. This work was in part supported by Key R&D Program of Zhejiang Province (No. 2023R5202) and Alibaba Research Intern Project.

References

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. 2010. Hedera: dynamic flow scheduling for data center networks.. In *Nsdi*, Vol. 10. San Jose, USA, 89–92.

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 503–514.

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

[4] Olivier Bonaventure, Christoph Paasch, and Gregory Detal. 2017. *Use cases and operational experience with multipath TCP*. Technical Report.

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

[6] Alibaba Cloud. 2023. A Detailed Explanation about Alibaba Cloud CIPU. https://www.alibabacloud.com/blog/599183. (2023). https://www.alibabacloud.com/blog/599183?spm=a3c0i.23458820.2359477120.3.76806e9bESi3SD

[7] Google Cloud. 2023. Introducing Falcon: A reliable, low-latency hardware transport. https://cloud.google.com/blog/topics/systems/introducing-falcon-a-reliable-low-latency-hardware-transport. (2023). Google Cloud Blog.

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

[9] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *2013 Proceedings IEEE INFOCOM*. IEEE, 2130–2138.

[10] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. 2013. *TCP extensions for multipath operation with multiple addresses*. Technical Report.

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

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

[13] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. 2014. 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*. 149–160.

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

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

[16] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. 2020. 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*. 514–528.

[17] Ming Li, Deepak Ganesan, and Prashant Shenoy. 2009. PRESTO: Feedback-driven data management in sensor networks. *IEEE/ACM Transactions on Networking* 17, 4 (2009), 1256–1269.

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

[19] Jianyuan Lu, Tian Pan, Shan He, Mao Miao, Guangzhe Zhou, Yining Qi, Shize Zhang, Enge Song, Xiaoqing Sun, Huaiyi Zhao, et al. 2023. Cloudsentry: Two-stage heavy hitter detection for cloud-scale gateway overload protection. *IEEE Transactions on Parallel and Distributed Systems* 35, 4 (2023), 616–633.

[20] Mostafa Abdulghafoor Mohammed and Nicolae Ţăpuş. 2023. A novel approach of reducing energy consumption by utilizing big data analysis in mobile cloud computing. *Mesopotamian Journal of Big Data* 2023 (2023), 110–117.

[21] Oracle Corporation. 2024. MySQL. https://www.mysql.com. (2024).

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

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

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

[25] Redis Ltd. 2024. Redis. (2024). [Online]. Available: https://redis.io/.

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

[27] Leah Shalev, Hani Ayoub, Nafea Bshara, Yuval Fatael, Ori Golan, Omer Ilany, Anna Levin, Zorik Machulsky, Kevin Milczewski, Marc Olson, et al. 2024. The Tail at AWS Scale. *IEEE Micro* (2024).

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

[29] Dian Shen, Bin Yang, Junxue Zhang, Fang Dong, and John CS Lui. 2024. eMPTCP: A Framework to Fully Extend Multipath TCP. *IEEE/ACM Transactions on Networking* (2024).

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

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

[32] Igor Sysoev. 2004. Nginx: An HTTP and Reverse Proxy Server. https://nginx.org. (2004). Accessed: 2025-02-28.

[33] Tesla. 2024. TTPoE: Tesla Transport Protocol over Ethernet. https://github.com/teslamotors/ttpoe. (2024). GitHub repository.

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

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

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

[37] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. 2012. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 139–150.

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

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

[40] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. 2021. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 418–432.