Lean: Saving OSS Network Cost by Application-Infrastructure Coordination

Qiankun Yue, Kun Qian*, Jiashi Lu, Xue Li, Qingchao Luo, Weibin Yan, Zhihong Li, Jianyao Sun, Junping Liang, Yuyao Chen, Liefeng Rong, Jing Zhang, Ensheng Xu, Boyu Ying, Fanghu Zheng, Guangren Chai, Erci Xu, Yuanchao Su, Dennis Cai
Alibaba Cloud

Abstract

Network is the single largest line item in a hyperscale Object Storage Service (OSS), consuming roughly 40% of an annual infrastructure bill measured in hundreds of millions of dollars. Two tiers dominate: the access network, where client traffic is locked into expensive in-region EdgePoPs despite 2–5× cheaper off-region bandwidth being widely available; and the intra-region network, where 3AZ replication amplifies every write into massive cross-AZ traffic that dominates the overall network bill.

We present Lean, a dedicated system that breaks through this barrier via application-infrastructure co-design, using SRv6 as a single unifying primitive to enable three capabilities: lightweight ingress units deployable at any EdgePoP, arbitrary traffic steering, and physical isolation of OSS traffic. In production for 4 years serving $O(1B)$ objects and $O(10K)$ enterprise customers, Lean has cut the OSS network bill by 55.6%, saving hundreds of millions of dollars annually.

CCS Concepts: • **Information systems → Cloud based storage**; • **Networks → Cloud computing**.

Keywords: Object Storage Service, Network Cost Reduction, Application-Infrastructure Coordination

ACM Reference Format:
Qiankun Yue, Kun Qian, Jiashi Lu, Xue Li, Qingchao Luo, Weibin Yan, Zhihong Li, Jianyao Sun, Junping Liang, Yuyao Chen, Liefeng Rong, Jing Zhang, Ensheng Xu, Boyu Ying, Fanghu Zheng, Guangren Chai, Erci Xu, Yuanchao Su, Dennis Cai. 2026. Lean: Saving OSS Network Cost by Application-Infrastructure Coordination. In *The 2026 ACM SIGOPS Annual Technical Conference (SIGOPS ATC 2026), November 15–18, 2026, Hong Kong, China.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3828523.3857056

*Corresponding author: Kun Qian (kunqian.qk@alibaba-inc.com).



This work is licensed under a Creative Commons Attribution 4.0 International License.

SIGOPS ATC 2026, Hong Kong, China
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2818-1/2026/11
https://doi.org/10.1145/3828523.3857056

1 Introduction

Layered abstraction lets a service scale fast; but coordinating across those layers lets the scaled service save.

Every major cloud runs an Object Storage Service (OSS) [2, 8, 13] that absorbs the bulk of its unstructured data: photos, model weights, training sets, backups, and logs. At hyperscale, a single provider's OSS sustains exabytes of capacity, tens of millions of requests per second, and an annual infrastructure bill measured in billions of dollars. The network is the single largest line item, at roughly 40% of that bill. A one-percent network saving is therefore worth tens of millions of dollars per year, making network cost the highest-leverage optimization target in all of OSS.

A cost paradox. A careful look at the OSS network bill reveals a paradox: cost runs inverse to physical distance. The bandwidth that crosses thousands of kilometers of long-haul fiber is the *cheapest* tier in our stack (~15.4% of the bill); the bandwidth that does not even leave the metro contributes significantly to the bill (~33.3%); and the access tier, which carries the smallest traffic volume, commands the highest cost. Two structural pathologies are responsible. First, at the edge, client traffic is locked into the most expensive BGP [32] transit even though much cheaper candidates exist. Second, inside a region, OSS can actually use only ~ 17.5% of the intra-AZ capacity it pays for: half is held back as failover headroom for dual-plane redundancy, and most of the rest must sit as advance buffer for the next expansion cycle: laying new fiber takes months, so the expansion-trigger watermark has to be set below the operational cap.

The straightforward fixes don't work. For the access tier, we deployed a cloud-relay scheme on ECS VMs in available regions, forwarding client traffic into OSS via the price differential, but the ECS compute bill exceeded the bandwidth saving, turning the optimization *cost-negative*. The first requirement is therefore **R1 (Any EdgePoP deployment)**: the forwarding logic must live inside the router itself, with no on-site compute. Once forwarding lives in the router, **R2 (Arbitrary traffic steering)** becomes inescapable: the application must steer client packets *to and from* a chosen entrance, reverse legs included, since naive designs lose responses to carrier-side anti-spoofing ACLs [35] along the default egress path.

For the intra-region tier, conventional QoS classes cannot fully prevent cross-business traffic interference; even after

we co-designed a dedicated *Silver* class with the network infrastructure team, production incidents showed that businesses sharing the same scheduler still block each other the moment capacity drops. The resulting uncertainty forces a conservative utilization watermark, leaving most provisioned bandwidth idle. The third requirement is therefore **R3 (Inter-business QoS isolation)**: OSS traffic must travel on *physically* disjoint resources so that isolation does not depend on a scheduler at all.

Both failures share the same root cause: *the bandwidth we want is already there, but the application has no way to reach it.* The conventional layered abstraction, where the application consumes the network as a black box, is precisely what prevents OSS from accessing the cheaper capacity that sits just beneath the API boundary.

Lean: application-infrastructure coordination. Our key insight is that unlocking stranded bandwidth requires *coordinated co-design* between the application layer and the network infrastructure: the application must be able to select ingress points, pin bidirectional forwarding paths, and claim physically isolated capacity, none of which the conventional network API exposes. We adopt *Segment Routing over IPv6* (SRv6) [10, 11], which encodes each forwarding instruction as a *Segment Identifier* (*SID*), as the enabling primitive: it is the thinnest mechanism that provides wide deployment (R1), source-routed path selection (R2), and customized traffic isolation (R3) at once, using only features already shipping in every network device.

On top of this primitive, Lean layers three integrated co-designs, one per requirement. The *Mini Core Backbone Router* (MCBR, §5.1) turns the smallest off-region site, often two routers in a leased rack with no on-site operations team, into a viable off-region EdgePoP by stripping the conventional Region PoP down to a stateless lights-out aggregator. *Arbitrary Traffic Steering* (§5.2) pins both the forward and reverse legs of a request to the chosen EdgePoP through a coordinated workflow spanning BGP, the Application Load Balancer, an OSS-kernel *TCP Option Address* (TOA) module, and a kernel-module SRv6 wrap on egress, sidestepping the carrier-side anti-spoofing ACLs that silently drop naive return paths. *Inter-business QoS Isolation* (§5.3) achieves physical isolation *without duplicating the network* by descending one layer below the conventional network abstraction: it steers OSS traffic onto a dedicated subset of 100 Gbps Egress Service Router (ESR) ports, a dedicated Optical Transponder Unit (OTU) [21] board, and a dedicated DWDM [20] wavelength block, all on *shared* data-comm (ESR) and OTN gear. The slogan is "shared facility, dedicated channel."

Deployment and impact. Lean is built and operated by Alibaba Cloud's OSS and network infrastructure teams. Lean has been in production for 4 years, serving the OSS service across 7 regions and 75 EdgePoPs. It has reduced the OSS-attributed network bill by 55.6%, saving hundreds of millions of dollars annually.

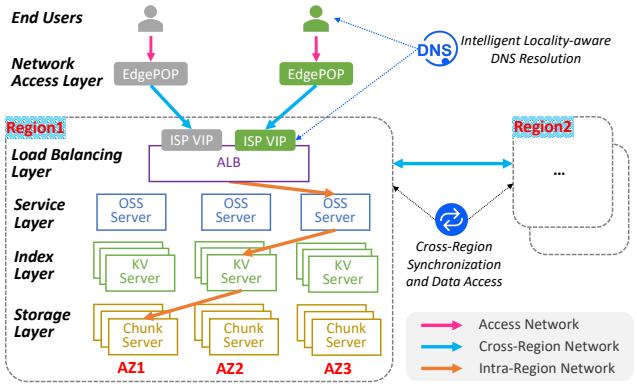


Figure 1. End-to-end access of OSS.

Contributions. This paper makes following contributions:

1. A counter-intuitive structural finding from our production cost analysis: in OSS, network cost runs *inverse* to physical distance, and the root cause is that large pools of already-paid bandwidth are stranded behind conventional layered abstractions (§3).
2. Lean, a deployed system that unlocks stranded bandwidth through *application-infrastructure coordination* (§4, §5). Three cross-product co-designs (OSS×EdgePoP for MCBR, OSS×ALB for Arbitrary Traffic Steering, and OSS×OTN for Inter-business QoS Isolation) each combine application-layer knowledge with a different infrastructure layer, exposing capabilities invisible at the standard API boundary. SRv6 serves as the common enabling primitive, introducing no new network node type.
3. Production deployment results at hyperscale (§6): a 55.6% reduction in OSS network cost, saving hundreds of millions of dollars annually.
4. Operational experience (§7), including lessons on generality across different clouds, global egress cost optimization, and a control-plane/data-plane consistency pitfall in traffic isolation.

2 Background

2.1 Accessing OSS

OSS is an archetypical cloud service [2, 8, 13] that provides scalable, durable, and cost-effective object storage to a wide range of applications. An outstanding feature of OSS is its geo-distributed access, which allows clients to read and write objects not just from cloud internal networks but also from the public Internet.

Figure 1 illustrates the end-to-end path of an OSS request. A client resolves the bucket URL through intelligent DNS, which directs the request to the nearest *EdgePOP* within its coverage area (e.g., a province) [28]. The EdgePOP forwards the request over the cloud's internal cross-region backbone to the target region, where a regional load balancer distributes it to an OSS server. The OSS server then looks up object metadata via the indexing layer (KV) and reads

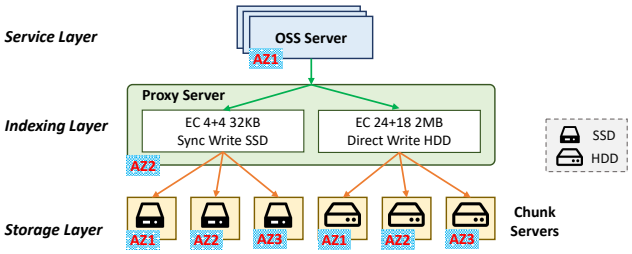


Figure 2. The 3AZ architecture of OSS.

Table 1. Network amplification of 3AZ OSS operations.

	Scheme	Amplification
OSS Write	3-Replica	2.67× (Same Region) 3× (Cross Region)
	EC	2.5× (Same Region) 2.83× (Cross Region)
OSS Read	Standard Read	0.66×

or writes data chunks from chunk servers that are spread across three availability zones (AZ1–AZ3).

2.2 Maintaining OSS

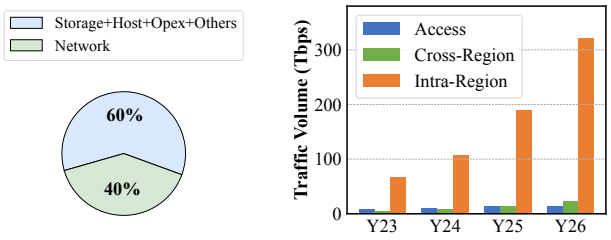
Like many storage service, OSS is required to provide strong durability guarantees. Major vendors (including ours) usually replicate data across three availability zones (3AZs) within a region (Figure 2). Each AZ has independent power, cooling, and networking, making correlated multi-AZ failures exceedingly rare. These properties yield "11–12 nines" of durability and 99.95–99.995% availability [2, 5, 8, 13].

However, 3AZ redundancy comes at the cost of significant **traffic amplification** as shown in Table 1. For writes, an OSS server in one AZ must distribute data chunks to servers in all three AZs. Due to differences in inter/intra-region traffic, the write amplification can be 2.67–3× under 3AZ replication [36], and 2.5–2.83× under erasure coding [19]. Fetching data does not introduce additional traffic. Table 1 counts bytes crossing AZs, not storage overhead. A write moves its data to a proxy once and to storage F times (Figure 2). With even placement each transfer crosses AZs with probability 2/3, giving $2(1+F)/3$, or $1+2F/3$ for ingress from another region, whose first hop always crosses. Replication has $F = 3$, and EC has $F = 2.75$: a 24+18 stripe plus the synchronous 4+4 write counted once. Note that OSS workloads are **write-dominated**, occupying 83.1% of the overall traffic. The dominance is an effect of amplification rather than the workload itself. As a result, the high write amplification of 3AZ redundancy is the primary driver of network cost.

3 Motivation

3.1 Understanding Network Costs in OSS

OSS's annual infrastructure bill can reach over billions of dollars, in which 40% goes to network (Figure 3a). That fraction is one of our largest costs and growth is fast as well.



(a) Cost distribution. **(b)** Network traffic volume.

Figure 3. Network traffic volume and cost distribution.

Table 2. Normalized traffic volume and cost.

	Traffic Volume	Overall Cost
Access Network	1	10
Cross-region Network	2.2	3
Intra-region Network	27.1	6.5

Specifically, intra-region traffic alone has increased by 6.25× over the past few years (Figure 3b), driven in large part by the rapid growth of AI training and inference workloads: checkpoints, model weights, and training datasets now account for a substantial and fast-growing share of intra-region storage I/O. At this scale, even a modest percentage reduction in network spending translates into saving tens of millions of dollars annually.

Table 2 shows that the overall network cost of OSS can be broken down into three categories, *Access Network* (i.e., the external-facing bandwidth between clients and EdgePoPs), the *Cross-region Network* (i.e., backbone between regions) and *Intra-region Network* (i.e., network among the three AZs within a region). Both columns are normalized to the access tier. The cost column reports OSS's share of our network's amortized capital and operating costs, billed internally as operating charges rather than a separate capital outlay.

Interestingly, the above three tiers reveal a paradox where the cost (dollars) runs in *inverse* proportion to distance (meters). The access tier carries only 1× of the traffic volume but is the most expensive per byte, because client traffic is locked into uncontrollable BGP. The cross-region tier, despite spanning thousands of kilometers of long-haul fiber and demanding the strictest reliability among all internal links, is the cheapest tier (only ~ 15.4% of network spend), because it runs over a *self-built* backbone whose amortized capex replaces the per-byte tariff of any external carrier (§3.3). Yet the same redundancy philosophy applied within a single region produces the opposite outcome: the intra-region tier carries 27.1× the volume at the cost share of 33.3%, with effective bandwidth utilization driven as low as 17.5% by reliability over-provisioning (§3.4).

The rest of this section examines three categories in turn: the access network (§3.2), the cross-region network (§3.3) and the intra-region network (§3.4).

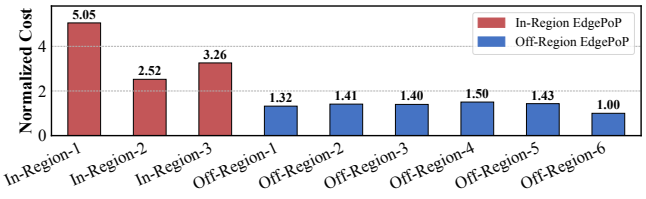


Figure 4. Access network cost in different EdgePoPs.

3.2 Access Network

The access network tier carries client traffic over the public Internet and is by far the most expensive on a per-unit basis. Internet Service Providers charge a uniform BGP transit price that is significantly higher than the unit cost of internal cloud bandwidth, as shown in Table 2. Moreover, the conventional access strategy is *nearest-entry*: intelligent DNS resolves each client to the geographically closest *in-region* EdgePoP (one co-located with an opened cloud region), minimizing unpredictable public-network hops and thereby optimizing end-user latency. A direct consequence is that the access-network cost is tightly coupled with the geographic distribution of users: regions with denser client populations naturally consume more expensive edge bandwidth.

However, as Figure 4 shows, using the public-Internet bandwidth directly from in-region EdgePoPs is neither the only option nor the cheapest. Across various provinces and areas, there exist abundant *off-region* bandwidth pools at sites with cheap static bandwidth but no cloud region presence, whose unit cost is 2–5× lower than that of in-region sites. These cheaper pools are hosted at off-region EdgePoPs scattered across the country; the catch is that they sit in locations too small to justify a full region-grade build, and conventional access architectures have no way to direct client traffic into them. The bandwidth is available and already paid for; what is missing is a minimal deployment unit that can turn each of these off-region sites into a usable ingress point.

Insight. The forwarding path between a client and an EdgePoP is determined hop-by-hop by BGP [32] across autonomous systems along the way. Neither the cloud provider nor the application can influence which EdgePoP the traffic enters, so clients are always routed to the nearest in-region EdgePoP, even when a 2–5× cheaper off-region alternative is nearby.

Fledgling solution: Cloud Relay. As a cloud-native application, we naturally explored whether existing cloud products could help bridge the cost gap. After an extensive survey, we identified a relay-based approach: by deploying lightweight proxy instances on Elastic Compute Service (ECS) virtual machines, client traffic can be first routed to these relay nodes and then forwarded over cheap internal links to the target OSS region, exploiting the in-region vs. off-region price differential shown in Figure 4.

We implemented this cloud-relay scheme and deployed it in production. Two structural limitations emerged.

Limitation 1: Limited cost-saving. A relay node's compute footprint forces it to run inside a region we have already opened. But the regions we have opened are, by selection, the ones with the highest cloud demand, and those are precisely the in-region sites where bandwidth tends to be most expensive (Figure 4). The relay can therefore only steer traffic onto an already-pricey subset of the candidate ingress locations; the cheaper off-region EdgePoP pools remain structurally out of reach, and the savings ceiling is still bounded.

Limitation 2: Poor performance. We deployed the scheme in production and ran the math. The result was unkind: the ECS compute bill for forwarding traffic ate the entire bandwidth saving, and then some, turning the optimization *cost-negative* overall (a detailed unit-economics comparison appears in Table 4). The mechanism behind this number is straightforward: ECS instances are general-purpose virtual machines not designed for high-throughput packet forwarding. Even with an aggressively optimized data path on a 4-core instance, each VM can only sustain at most 15 Gbps of relay throughput; scaling to full access traffic (typically 500-600 Gbps) would require dozens of instances, whose compute cost exceeds the bandwidth saving by the margin above. Kernel bypass could lower relay cost, but ECS remains unavailable at cheaper EdgePoPs without a cloud region.

Ideal solution for access network. The analysis above points to two requirements:

[R1] Any EdgePoP deployment. The solution must be deployable at *any* EdgePoP, including the smallest EdgePoPs that may consist of only one or two racks without a full datacenter facility. Only by reaching all available EdgePoPs can we fully exploit low-cost bandwidth.

[R2] Arbitrary traffic steering. Despite its efficiency shortcomings, the cloud-relay approach demonstrates a valuable property: a fully software-defined forwarding plane enables completely customizable routing decisions. An ideal solution must preserve this ability to perform application-level, cost-aware traffic steering while eliminating the per-instance throughput bottleneck.

3.3 Cross-region Network.

In contrast to the access tier, the cross-region network, which connects geographically distributed cloud regions through a long-haul optical backbone, is the cheapest tier in the entire stack, contributing only ~ 15.4% to the OSS network bill (Table 2) despite spanning thousands of kilometers and meeting the strictest reliability targets among all internal links. We include this tier in the discussion not as an open optimization problem but as an instructive counter-example: if strict reliability can be provided cheaply across regions, why is the intra-region network so costly?

The reliability architecture. The bedrock of the cross-region tier is a *Dual-Plane Triple-Route* redundancy architecture (Figure 5), designed to absorb the single-cable failures that are far from rare in operational practice. Municipal construction accidents that sever fiber, pole collapses caused by natural disasters, and similar hazards are ever-present.

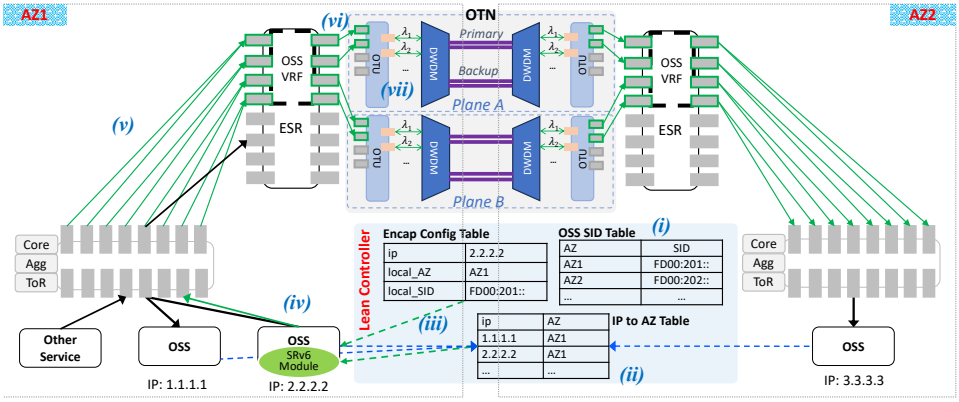


Figure 5. OSS intra-region end-to-end network architecture.

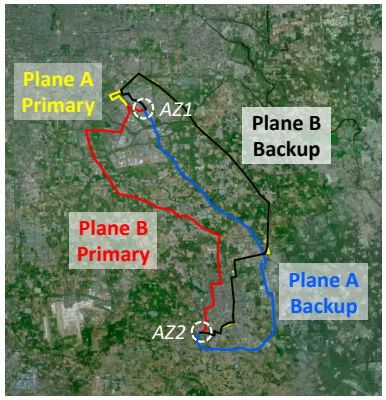
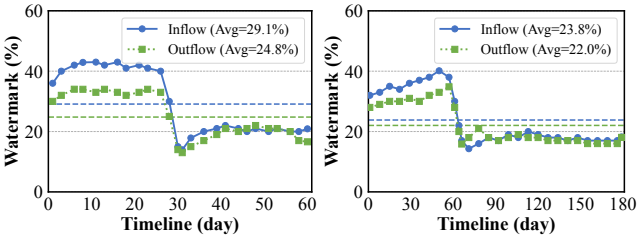


Figure 6. Path separation.



(a) Region-1. **(b)** Region-2.

Figure 7. Typical flow trends and capacity expansions.

To this end, two transport planes (Plane A and Plane B) each deploy an independent cluster of Optical Transport Network (OTN) [21] equipment with fully separate optical amplifier chains and Dense Wavelength-Division Multiplexing (DWDM) [20] channels. Plane A is further provisioned along two geographically disjoint fiber routes (*e.g.,* one along a southern corridor and the other along a northern corridor, as illustrated in Figure 6), so that end-to-end three physically independent optical paths are formed in total. This design provides *dual-failure tolerance*: even when any two of the three optical routes are simultaneously cut, traffic can still be carried on the remaining route without service interruption.

Why redundancy stays cheap here. The decisive factor is that the cross-region backbone is *self-built*: rather than purchasing wholesale bandwidth from telecom carriers at a per-byte tariff, Alibaba Cloud owns and operates the underlying long-haul infrastructure end-to-end: dark fiber, right-of-way, OTN equipment, and DWDM channels. The cost paid per delivered byte is therefore the amortized capital and operating cost of this infrastructure.

3.4 Intra-region Network.

Counter-intuitively, the intra-region network, physically the shortest tier in the entire stack, is a significant cost contributor, accounting for roughly 33.3% of the total OSS network spending (Table 2). The arithmetic underneath is brutal: in a typical region, OSS pays for 38.4 Tbps of provisioned intra-region capacity but uses only 6.72 Tbps of it.

Why the cost is so high. Two factors compound to drive up the bill. The first is intrinsic to OSS itself: as discussed in Section 2.2, every object is replicated across three AZs, so a write amplification of up to 3× lands almost entirely on cross-AZ links. The second, and more surprising, factor is that the intra-region network itself is heavily over-provisioned along *two orthogonal dimensions* to satisfy stringent reliability requirements.

(a) Dual-plane low-watermark redundancy. As in the cross-region backbone, cross-AZ traffic is carried by two physically independent planes (Plane A and Plane B), whose equipment, optical paths, and fiber routes are kept strictly disjoint. The operational watermark on each plane is then capped at 50% of its provisioned capacity, so that even after a worst-case incident such as a severed fiber or a major router failure, the surviving plane alone can absorb the entire traffic.

(b) Coarse capacity-expansion granularity. On top of the 50% runtime cap, lead-time considerations push the effective watermark substantially lower (Figure 7). A single optical fiber has a hard physical capacity limit (*i.e.,* 19.2 Tbps), and any expansion requires civil construction, new equipment installation, and the laying of additional fiber, an end-to-end process that takes months. To preserve enough lead time, the expansion-trigger watermark is therefore set at only 35%. For example, starting from a 19.2 Tbps deployment, OSS must request an expansion once its regional bandwidth usage reaches 6.72 Tbps; the expansion doubles the total capacity to 38.4 Tbps, dropping the effective watermark to a mere 17.5%. Because the expansion is attributed to OSS, the amortized cost of the entire newly built capacity is charged back to OSS, even though the bulk of that capacity now sits idle as reliability headroom rather than serving OSS traffic.

The combined effect is striking: a substantial fraction of the intra-region cost attributed to OSS does not pay for bytes actually delivered by user traffic, but for bandwidth that sits idle as a reliability buffer for dual-plane redundancy and the next expansion cycle.

Insight. The key observation is that OSS itself already provides AZ-level fault tolerance at the application layer through

3AZ replication: when a cross-AZ path between two AZs goes down, the remaining replicas keep the service available. OSS therefore does not actually need the highly fault-tolerant, guaranteed-delivery service that the underlying infrastructure provides; a *best-effort* intra-region forwarding service would already match its true reliability profile, and at a fraction of the cost. The remaining question is how to grant OSS access to that idle, lower-reliability bandwidth without forcing it to pay for the full reliability guarantee.

Fledgling solution: QoS-based differentiation. The most natural answer is to differentiate via QoS classes [6]. The infrastructure network team had already provisioned two priority levels, *Gold* (high) and *Copper* (low), sharing each link via Weighted Round Robin (WRR) at a 40:1 ratio.

Unsuccessful attempt 1: All OSS traffic on Copper. The simplest approach was to remap all OSS traffic to the Copper class. In production, however, this produced heavy packet loss and an effective throughput far below expectation. Two structural issues compounded. First, Copper is the first class dropped under any network incident, so its capacity is unbounded from below. Second, Copper is already heavily consumed by other business lines whose demand we cannot bound from above. Meanwhile, OSS front-end traffic cannot tolerate the disordered packet loss that Copper produces, so the class is fundamentally mismatched with OSS's reliability profile.

Unsuccessful attempt 2: A new Silver class. We then worked with the infrastructure network team to introduce an intermediate *Silver* class. After extensive testing, the WRR ratio was tuned to 40:8:1 for Gold:Silver:Copper, a balance between giving Silver enough share to absorb OSS's traffic and keeping it small enough not to starve Gold under contention.

Even this carefully calibrated scheme fell short in production. The failure mode was cross-business interference *within* the Silver class itself. An illustrative incident: on a 100 Gbps link, business A was sending 35 Gbps of Silver traffic; business B was sending 30 Gbps. A failure halved the link. *Both* businesses started losing packets, even business B, which was nowhere near its share. The QoS class promised fairness when there was enough to go around; the moment there was not, it delivered neither fairness nor isolation.

Ideal solution: [R3] Inter-business QoS isolation. Every QoS scheme we tried sliced the same shared pool more finely. But the interference came from the *sharing*, never from the slicing. Both attempts above failed at the same point: the moment a failure shifted demand, businesses in the same Silver (or Copper) queue blocked each other, regardless of how carefully their nominal allocations had been calibrated. The missing primitive is not a finer class but a *wall*: a forwarding resource on which OSS's bytes travel and that no other business can address.

The ideal solution should deliver the deterministic resource isolation for OSS while continuing to amortize the capex of the existing shared facility across all businesses.

4 Basic Idea

Key insight: coordinate, don't just consume. Our central observation is that the stranded bandwidth cannot be unlocked by improving any single layer in isolation. No amount of application-level cleverness can bypass BGP's routing decisions; no network-level QoS weight can substitute for physical separation. The savings come only when the application and the infrastructure *jointly* expose and exploit structure that neither side can access alone.

This "application-infrastructure coordination" principle materializes as three co-designs, each fusing OSS's application-layer knowledge with a different infrastructure layer. We adopt SRv6 [10, 11] as the unifying primitive because it is the thinnest mechanism that delivers all three capabilities at once, using only features already shipping in our production routers with no new node type introduced.

- **R1: Any EdgePoP deployment (OSS × EdgePoP, §5.1).** The cheap bandwidth is scattered across dozens of off-region sites too small for a conventional Region PoP. Because SRv6 forwarding is a native router function, OSS can turn each such site into a usable ingress point without deploying any on-site compute.
- **R2: Arbitrary traffic steering (OSS × ALB, §5.2).** Reaching off-region sites is useless unless return traffic can be pinned back through the same entrance. By coordinating across the load balancer, the OSS kernel, and SRv6 source routing, the system steers both forward and reverse legs through any chosen EdgePoP.
- **R3: Inter-business QoS isolation (OSS × OTN, §5.3).** Scheduler-based QoS repeatedly failed because the underlying resource was still shared. By coordinating with the optical-transport layer, OSS obtains a physically dedicated forwarding channel on shared infrastructure, which we call "shared facility, dedicated channel."

SRv6 as the unifying primitive. SRv6 [10, 11] is a standardized network-layer service differentiation mechanism that makes all three co-designs deployable on existing infrastructure: every router executes SRv6 natively in its hardware fast path (**R1**); cloud products such as ALB can encode and consume SRv6 segments through a standard IPv6 extension header (**R2**); and every switch in the forwarding chain (ToR, aggregation, and core) can correctly identify and forward SRv6-marked traffic onto dedicated physical resources (**R3**).

5 Design

5.1 Any EdgePoP Deployment

Satisfying **R1** (an ingress unit deployable at *every* off-region EdgePoP offering low-cost static bandwidth, without on-site compute) required us to replace the conventional "one Region PoP per in-region site" model with a much lighter unit purpose-built for off-region cheap-bandwidth aggregation. We call this unit the *MCBR*.

Table 3. Feature comparison between Region PoP and MCBR.

	Region PoP	MCBR
Scrubbing	Self-built	Carrier-side
DNS	✓	✗
Routing	✓	✓
En-/De-cap	✓	✓
Size	36U	4U
Cost	100	1



Figure 8. The MCBR device.

MCBR design. The legacy solution is relying on a feature-complete unit bundling a chassis-based Core Backbone Router (CBR), a self-built DDoS scrubbing cluster, an Anycast DNS resolver, a cloud network access gateway, and private-line aggregation, totaling 36U of rack space (Table 3). An off-region site, however, has a single mission: aggregate local public-Internet bandwidth and backhaul it to our cloud.

As Table 3 shows, the MCBR retains only the essential functions, and offloads DDoS scrubbing to the upstream carrier rather than hosting it on-site, eliminating DNS, cloud network access, and private-line aggregation entirely. The result is a 4U footprint (Figure 8) deployable at any off-region site, including sites that consist of only one or two racks with no datacenter facility, at a 100× cost reduction.

Backhaul to the Region runs over a private Optical Transport Network (OTN) provisioned in the same *Dual-Plane Triple-Route* topology described in §3.3. An MCBR terminates no application sessions and holds no per-flow state of its own; it is, by design, a stateless aggregation device. This minimal control-plane and operations footprint is what makes the MCBR deployable at arbitrarily many off-region EdgePoPs, closing the gap that **R1** (§4) demanded: every off-region EdgePoP with a favorable static-bandwidth tariff is now a usable ingress point, at a marginal cost that is a small fraction of opening a Region PoP.

5.2 Arbitrary Traffic Steering

The Any EdgePoP design of §5.1 solves the deployment problem: off-region sites can now accept client traffic. However, deployment alone is not enough: the *return path* must be actively managed. Each carrier enforces anti-spoofing ACLs [35] that only permit packets whose source address matches the local EdgePoP's advertised prefix; responses that exit through a different EdgePoP are silently dropped, rendering the service unavailable.

This subsection describes how we achieve arbitrary traffic steering that pins both the forward and reverse legs of a request to the same EdgePoP (Figure 9).

Client to ALB: finding the correct entrance. Each EdgePoP is associated with a dedicated EdgePoP-anchored Virtual IP (*vip*) range that fronts the OSS service. Before a *vip* range goes live, capacity for it is provisioned on the ALB cluster behind that EdgePoP, and ALB begins advertising the corresponding route; ordinary BGP carries the advertisement out through the EdgePoP and the public Internet re-learns it hop by hop. When a client resolves the OSS service name, DNS returns one of these EdgePoP-anchored *vip*s as the destination address. The client's packet therefore enters the public Internet with a destination that is, by construction, already pinned to a specific EdgePoP, and standard BGP forwarding delivers it to that EdgePoP, which hands it on to the correponding ALB instance.

ALB to OSS server: cross-product integration. At ALB the destination address *must* change: the packet has to reach the actual OSS server, which is reachable only via an internal address, not the public *vip*. A conventional load balancer would rewrite the destination and discard the original. That discard is precisely what breaks **R2**. Once the packet arrives at the OSS server, it can no longer tell which EdgePoP it entered through. We solve this by treating ALB and OSS as a single, jointly designed pipeline rather than two opaque products.

(i) ALB stashes the vip in a TCP option. When ALB rewrites the destination address from the public *vip* to the OSS server's internal address, it does *not* simply throw the *vip* away. It copies the original destination into a reserved TCP option field on the same segment, so the rewrite preserves the ingress identity instead of erasing it.

(ii) OSS recovers the vip via TOA. On the OSS server, the TOA kernel module parses the TCP option on connection establishment and exposes the original *vip* to the OSS business stack. At response time, the server emits a packet whose destination is the client IP (*cip*) and whose source is the *vip* advertised by ALB, restoring the exact address pair the client first observed.

OSS server to client: returning through the correct exit. Restoring the source address in step (ii) is necessary but not sufficient. The response will only survive the carrier-side ACLs if it actually egresses through the EdgePoP that advertised that source address. The OSS server now holds the correct address pair but, speaking at the application layer, has no way to dictate which physical EdgePoP the network uses on the way out. We bridge that final gap by lifting the ingress *vip* one more time, from in-stack metadata into an SRv6 segment in the IPv6 header, so the underlay can route on it directly.

(iii) OSS server encapsulates the response in SRv6. On egress from the OSS server, a kernel-module hook consults the *vip*→*SID* table (each *SID* encodes the exit EdgePoP) populated by the Lean controller and encapsulates the packet with an SRv6 header whose *SID* identifies the originating EdgePoP (Figure 10). The *vip* encodes *which* EdgePoP to exit through, and the SRv6 header translates that intent into a routable segment that the underlay can act on directly.

(iv) ESR forwards based on SRv6 SID. At the ESR, the SRv6 *SID* is resolved to the matching forwarding entry, and the

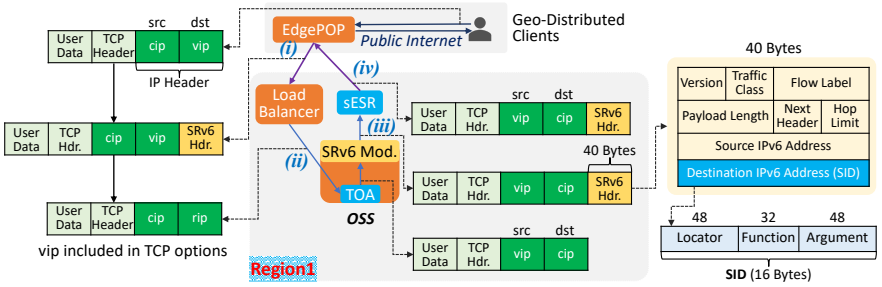


Figure 9. End-to-end arbitrary traffic steering.

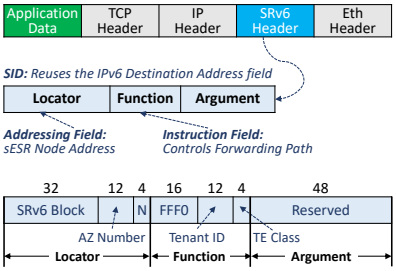


Figure 10. The SRv6 packet format.

packet is steered out through the same EdgePoP the request originally entered.

Why this integration scales. The forward path relies on native BGP; the return path is steered entirely by an SRv6 header attached at OSS egress. Nothing in between holds per-flow state, and the customization footprint is confined to three small touchpoints: a TCP-option write at ALB, a TOA parse at the OSS server, and a SRv6 wrap kernel-module.

5.3 Inter-business QoS Isolation

The two failed QoS attempts in §3.4 share a single structural root: both partition a *single shared pool* into ever finer service classes (Gold, Silver, Copper), and rely on a scheduler to mediate access among businesses. As long as the pool is shared, however, no scheduling weight can prevent one business's burst from spilling into another's. The missing primitive is therefore not a finer class; it is *physical separation between OSS and other business.*

A naive reading of "physical separation" would suggest provisioning OSS its own routers, OTN frames, and fiber, in effect a parallel private network whose cost would erase every saving we set out to capture. The crux of our design is that physical separation can be achieved *without* duplicating the facility. We worked with our network infrastructure team to take apart the full forwarding chain end to end, and isolated two points where the hardware exposes enough internal granularity for OSS to own a dedicated slice: the *entrance* (the ESR uplink) and the *transmission* (the data-comm/OTN stack between AZs).

Entrance: from one logical node to independent physical ports. The standard mental model treats the ESR as a single logical node: every business stream that arrives funnels into the same forwarding engine and is multiplexed onto the same uplink, with WRR weights being the only knob that separates them, making cross-business interference unavoidable.

The model is, however, an abstraction. As shown in Figure 5, a production ESR is in fact a chassis exposing 256 functionally-equivalent 100 Gbps physically independent uplink ports. We leverage SRv6 to exploit this physical structure for isolation. A lightweight control-plane pipeline tags every cross-AZ OSS packet with an SRv6 *SID*:

(i) SID acquisition. The Lean controller pulls the $\langle AZ, SID \rangle$ mapping and stores it as the *OSS SID Table*, which maps each AZ to the SRv6 SID.

(ii) Server registration. Each OSS server periodically heartbeats its own $\langle IP, AZ \rangle$ tuple to the Lean controller, building a global *IP to AZ Table* that records which AZ every server belongs to.

(iii) Encap Config generation. The Lean controller joins the two tables to produce a per-server *Encap Config Table*, and pushes both the IP to AZ Table and the Encap Config Table to the SRv6 kernel modules on every OSS server.

(iv) Packet encapsulation. On the egress data path, the SRv6 module looks up the destination IP in the IP to AZ Table to obtain the destination AZ. If the destination AZ equals the *local_AZ*, the packet is forwarded without encapsulation; otherwise, the module wraps the packet in an SRv6 header using the destination AZ's SID.

Transmission: isolation across data-comm and OTN. An OSS packet still has to cross the same data-comm boxes, the same OTN frames, and the same long-haul fiber that carry every other intra-region business. We therefore extend the same "dedicated slice on shared facility" principle down three further layers of the transmission stack, each layer chosen because it exposes a hardware primitive fine-grained enough to slice without provisioning a parallel facility:

(v) Dedicated port groups. On the ESR, a dedicated *OSS Virtual Routing and Forwarding (VRF)* matches arriving packets by their SRv6 *SID* and hashes them to a reserved subset of the 256 uplink ports, so OSS and non-OSS traffic part company at the first forwarding decision, physically separating them from that point onward.

(vi) Dedicated OTU boards in dedicated chassis slots. On the OTN side, each OSS uplink lands on its own Optical Transponder Unit (OTU) [21] board, occupying a reserved slot of the OTN chassis with its own client-side gray-light ports and its own line-side coherent modules.

(vii) DWDM wavelength slots. At the optical layer, OSS is assigned a contiguous block of DWDM wavelength slots on the ITU-T G.694.1 frequency grid [20] (*e.g.*, λ_1–λ_2), strictly disjoint from the wavelengths that carry shared-business traffic (*e.g.*, λ_3). Two services may now share the same fiber cable while occupying disjoint photons.

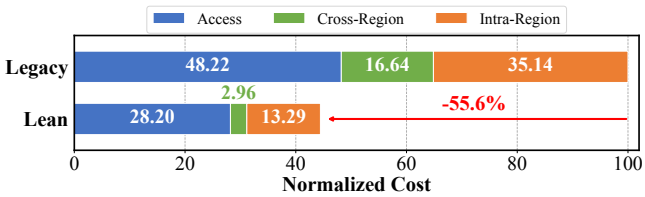


Figure 11. Network cost reduction of Lean in 2026.

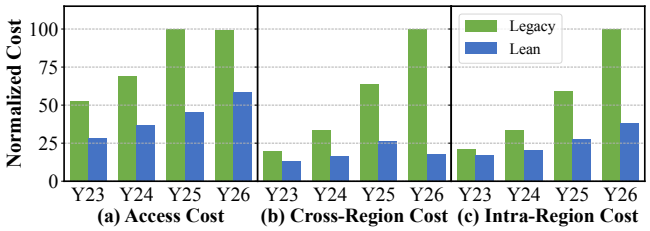


Figure 12. Normalized network cost across three tiers.

Composing the three layers yields a property that the QoS schemes of §3.4 could not deliver: an OSS packet traverses dedicated data-comm ports, a dedicated OTU board, and a dedicated wavelength from entrance to egress, without any additional equipment cost.

A natural concern is the bandwidth stranded by dedicating physical ports, and the 256-port granularity is what makes this negligible. Each port carries 1/256 of the ESR's uplink, roughly 0.39% of total capacity, so reserving a port count that matches OSS's actual share of the regional bandwidth wastes essentially nothing in practice.

Why this satisfies R3 where QoS could not. We summarize the property as "*shared facility, dedicated channel.*" OSS reuses the same infrastructure as every other business. But the packets themselves travel on a slice of that facility that no other business can address. The bandwidth guarantee OSS enjoys is therefore determined by physics, not by a scheduler, and is consequently equivalent to what a fully-private parallel network would provide. This is precisely the business-level isolation that **R3** demanded.

Determining the headroom under physical isolation. Inter-business isolation guarantees that OSS traffic never contends with other businesses, but OSS's own capacity must still be right-sized. We outline how to compute a safe operating watermark systematically. In this part, we use write traffic as the running example (the watermark calculation for read traffic follows the same logic).

In the fault-free state, the total intra-region write traffic is D. Because the three AZs split the load evenly, each AZ's egress demand is $1/3D$. After inter-business isolation the underlying network no longer provides redundancy protection for OSS; we must therefore dimension the dedicated channel to absorb an AZ failure. When one AZ goes down the surviving two AZs absorb the full load, raising per-AZ egress to $1/2D$ (a 50% increase). To ensure the channel is never overloaded post-failure, the normal-case utilization must stay below 66.6%. In practice we also reserve headroom for background tasks such as garbage collection and data rebalancing. In production we therefore set the operating watermark at 50%.

The limit generalizes into a simple bound on the balance between cost and reliability. With N AZs that must tolerate f failures and load spread evenly, the surviving AZs must absorb the demand of the lost ones, which keeps the utilization watermark W at or below $(N-f)/N$; our deployment is $N = 3$, $f = 1$, hence the 66.6% ceiling above. Once demand and unit prices are fixed, the capacity OSS pays for scales as $1/W$, so the watermark, not the traffic volume, is what determines the intra-region bill.

6 Evaluations in Production

Lean serves Alibaba Cloud's OSS across all regions for 4 years, covering 75 EdgePoPs across 7 regions, supporting $O(10)$Tbps bandwidth, $O(1B)$ objects, and $O(10K)$ enterprise customers. We first present the overall cost saving achieved by Lean in production (§6.1), and then drill into the three design pillars of §5: Any EdgePoP deployment (§6.2), Arbitrary Traffic Steering (§6.3), and Flexible QoS (§6.4).

6.1 Cost Saving in Production

Overall cost reduction. Figure 11 reports the overall network cost reduction in production. Measured against the legacy system at 2026 production traffic volumes, Lean reduces total network cost by **55.6%**, with Access Network costs down by 41.5% and Intra-Region costs down by 62.2%. Cross-region costs also decrease owing to the extension of inter-business QoS isolation to the cross-region domain. Legacy is the network we would run today without Lean, so none of the saving comes from traffic growth, and both follow the accounting of Table 2.

Each tier maps to one pillar. The access saving comes from **R1** and **R2** moving ingress onto off-region EdgePoPs whose bandwidth costs 2–5× less (Figure 4). The cross-region cost falls the most in relative terms, but from the smallest base. The intra-region saving combines two effects: isolation lifts usable utilization by 1.8× (Figure 16), which removes about 44% of the provisioned capacity, and the balance is expansion headroom that §3.4 charged to OSS but the dedicated channel no longer needs.

In total, Lean saves hundreds of millions of dollars annually for our OSS business.

Cost reduction in different tiers. Figure 12 compares the legacy network cost (without Lean) against the optimized cost (with Lean) for each of the three tiers (access, cross-region, and intra-region) from 2023 to 2026. The access tier shows a widening gap as more off-region EdgePoPs come online and absorb traffic that would otherwise flow through expensive in-region EdgePoPs. The intra-region tier shows the most dramatic divergence, consistent with the physical isolation reclaiming over-provisioned headroom that would otherwise grow linearly with capacity expansion.

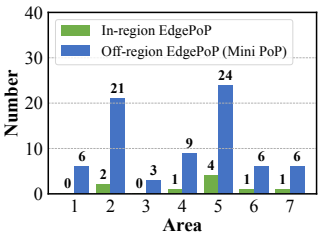


Figure 13. Number of different EdgePoPs in each area.

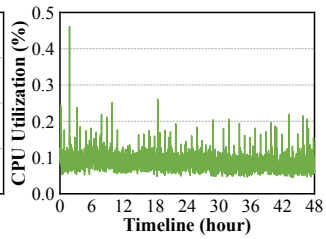


Figure 14. Lean controller CPU utilization.

Table 4. Per-device capability and unit cost.

	ECS Relay (VM)	MCBR
Throughput / device	15 Gbps	6.4 Tbps
CapEx / device	$5700	$7100
All-in cost / Tbps	$380K	$1.1K

6.2 Any EdgePoP Deployment

The Any EdgePoP design (§5.1) makes two quantifiable claims: that an MCBR can turn off-region sites, where deploying a full in-region Region PoP is uneconomical, into usable off-region EdgePoPs, and that it can carry production-scale ingress at a small fraction of the per-bit cost of the ECS-relay alternative we rejected in §3.2.

Off-region EdgePoP vs. in-region Region PoP. Figure 13 compares the number of off-region EdgePoPs against in-region Region PoPs, broken down by geographic area. Across all seven areas, 75 off-region EdgePoPs have been deployed versus 27 in-region Region PoPs, a 2.8× expansion of the overall ingress footprint. The gain is most pronounced in areas where cheap static bandwidth is abundant but cloud-region demand is too low to justify an in-region deployment. Deploying an off-region EdgePoP also takes 1–3 months, against 6–12 months for an in-region Region PoP.

MCBR vs. ECS relay. Table 4 compares an MCBR to the ECS-relay data path of §3.2 along three axes. A single MCBR sustains 6.4 Tbps, over 400× the 15 Gbps ceiling of a typical ECS relay VM, while their capital costs are in the same ballpark ($7100 vs. $5700). The resulting all-in cost per Tbps of relay capacity is $1.1K for an MCBR versus $380K for the ECS path, a 345× reduction that turns a cost-negative scheme into a substantial net saving.

Latency of off-region ingress. Requests steered to an EdgePoP outside the target region and backhauled over our backbone do not inflate latency. For a typical cross-region access pattern in production, request latency under Lean is 25.3/45.1 ms at p50/p99, against 27.8/55.8 ms under Legacy. The gain comes from replacing a long carrier segment with our own backbone, which is faster and more predictable.

6.3 Arbitrary Traffic Steering

The central question in this part is whether the state and compute footprint introduced by our solution grows slowly enough to support arbitrarily many EdgePoPs. We evaluate four measurements covering state, control-plane load, and data-plane overhead.

Table 5. ALB routing-table size.

	Reg.1	Reg.2	Reg.3	Reg.4	Reg.5	Reg.6
Table Size	235	174	140	133	26	66
Percent (%)	11.75	8.70	7.00	6.65	1.30	3.30

Table 6. Consumed number of SRv6 *SID*s.

	Reg.1	Reg.2	Reg.3	Reg.4	Reg.5	Reg.6
#*SID*	72	57	45	42	24	24
Percent (%)	0.11	0.09	0.07	0.06	0.04	0.04

ALB routing-table size. Table 5 reports the number of routing-table entries an ALB instance maintains for the EdgePoP-anchored *vip* ranges, broken down by regions. As expected, the table size grows linearly in the number of EdgePoPs covered by each ALB. Across the six representative regions, the largest ALB (Region-1) holds only 235 entries, accounting for merely 11.75% of the available routing-table capacity, while the smallest one (Region-5) uses just 26 entries (1.3%). The total footprint across all six regions stays well below 12% of any single ALB's capacity, leaving ample headroom for adding many more EdgePoPs.

IPv6 address-space cost of scaling. A subtler scaling cost is the IPv6 prefix consumed by SRv6 *SID*s. Table 6 reports, for the same six representative regions, the number of *SID*s allocated and the corresponding fraction of the assigned IPv6 segment consumed by the *SID* encoding. Even in the largest region (Region-1) only 72 *SID*s are allocated, occupying merely 0.11% of the available IPv6 segment. The average IPv6 footprint stays well under 0.5% of any single segment.

Lean controller CPU utilization. Figure 14 plots the CPU utilization of the active Lean controller over a representative 2-day window. Throughout the 48-hour observation period, CPU utilization remains below 0.5%, with the vast majority of samples staying under 0.1%. This confirms that the Lean controller is far from being a bottleneck and leaves ample headroom for scaling.

Data-plane overhead: SRv6 encapsulation. The encapsulation prepends a 40-byte IPv6 header to each outgoing IPv4 packet (standard IPv4-over-IPv6 tunneling, no SRH extension header) and the in-kernel encapsulation cost stays below 10 μs, negligible against the ~1.5 ms cross-AZ RTT. The metric that actually matters in production is its *target serving throughput*.

OSS delivers two server tiers in production. *Standard* servers are equipped with Mellanox ConnectX-4 (CX4) NICs and targeted at 20 Gb/s of serving throughput per server, while *Premium* servers use Mellanox ConnectX-6 (CX6) NICs and targeted at 40 Gb/s. Each tier's planned throughput is jointly determined by disk performance, memory size, CPU core count, and the OSS software stack. Table 7 reports single-core throughput across the two tiers.

Table 7. Single-core throughput under SRv6 encapsulation.

Tier / NIC	Target (Gb/s)	Configuration	Tput (Gb/s)	vs. target
Standard / CX4	20	SRv6, MTU 1500	17.6	−12%
		SRv6, Jumbo 7000	30.4	+52%
Premium / CX6	40	SRv6, MTU 1500	48.0	+20%
		SRv6, Jumbo 7000	48.8	+22%

Standard tier (CX4). Naively enabling SRv6 with the default 1500 B MTU drops throughput to 17.6 Gb/s, which fails the delivery bar. The root cause is that CX4 cannot actually perform TCP Segmentation Offload (TSO) on IPv4-over-IPv6 encapsulated packets despite reporting TSO as enabled. To mitigate this, we deploy jumbo frames (MTU = 7000): enlarging the per-packet payload reduces software-segmentation calls by ∼80%, recovers throughput to 30.4 Gb/s.

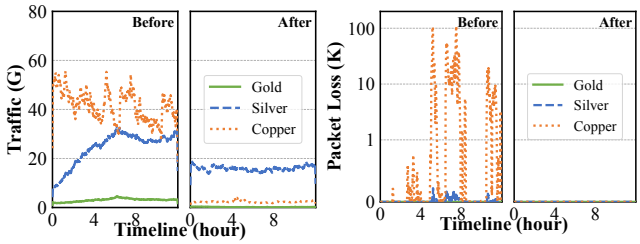
Premium tier (CX6). CX6 recognizes the encapsulation format and performs hardware TSO normally, leaving software checksum as the only remaining per-packet cost. As a result, every configuration in Table 7 comfortably exceeds the 40 Gb/s targeted throughput.

Operational complexity. Routing entries and *SID*s grow with the number of EdgePoPs and AZs, and server registrations with the number of servers, but none of this state grows with flows or customers. No SRv6 policy is written by hand, since the controller joins the *AZ*→*SID* and IP→*AZ* tables and generates each server's encapsulation configuration automatically (§5.3). The complexity we did pay is organizational: moving the OSS server fleet to dual-stack IPv4/IPv6, agreeing on the *SID* scheme across teams, and bringing each new EdgePoP into service.

6.4 Inter-business QoS Isolation

We evaluate the Inter-business QoS design (§5.3) along two dimensions: whether the isolation eliminates cross-business interference and whether it lifts effective OTN utilization.

Cross-business interference: before vs. after isolation. Figure 15 illustrates the effect of Lean's QoS isolation on a production cross-AZ link. Before Lean, OSS traffic was assigned the Copper class and consumed the majority of the link's capacity (Figure 15a, left). When Silver-class traffic surged, the aggregate demand exceeded the link's serviceable bandwidth; both the Silver and Copper queues suffered heavy packet loss (Figure 15b, left). The incident damaged OSS throughput and, worse, disrupted the more SLO-sensitive Silver business, exactly the cross-class interference that the Gold/Silver/Copper WRR scheme of §3.4 was supposed to prevent. After Lean is deployed, OSS traffic is physically isolated from all other businesses, and experiences zero packet loss (Figure 15b, right). At the same time, removing OSS from the shared pool lowers the overall utilization for the remaining businesses, so the Silver class likewise sees no loss, a gain for both sides that scheduler-based QoS could never deliver.



(a) Network traffic. **(b)** Packet loss.

Figure 15. Case study for inter-business QoS isolation.

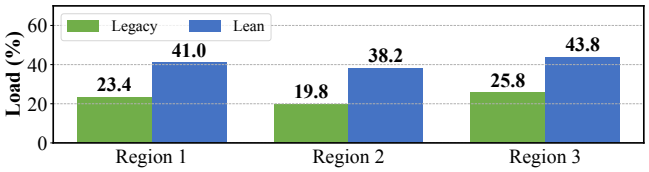


Figure 16. Intra-region network utilization.

OTN utilization headroom recovered. Figure 16 compares the effective OTN utilization on cross-AZ links before (Legacy) and after (Lean) the QoS-isolation rollout across three representative regions. Under the legacy shared-QoS regime, utilization ranges from 19.8% to 25.8% because the dual-plane failure-tolerance budget forces a conservative watermark. After isolation, utilization rises to 38.2–43.8%, roughly a 1.8× improvement, because the recovered capacity comes from a dedicated slice that OSS owns, not from the shared scheduling pool that the dual-plane budget protects.

7 Experience

7.1 Generality of Lean

The experience we most want to share is that the design of Lean is intentionally general-purpose: every building block relies on commodity hardware, standardized protocols, or cross-team integration patterns that are reproducible in any major cloud provider. Our key takeaway is that cross-layer coordination within a cloud, where application teams and infrastructure teams co-design interfaces, yields substantial cost and performance gains that compound across every pillar of the system.

Three principles behind these mechanisms travel beyond our deployment. First, do not pay for fault tolerance twice at different layers: OSS already tolerates the loss of an AZ, so holding network-level capacity to survive the same event duplicates protection the application never asked for. Second, provision the network to the failures the application already tolerates, not to those a generic service might see; this is what turns reserved headroom into billable saving. Third, when a shared scheduler cannot promise predictable service, escalate from QoS classes to physical isolation while reusing the facility underneath.

MCBR. The MCBR definition captures the minimal capability set of an access device. Our reference deployment fits in the 2U+2U commodity-off-the-shelf appliance, meaning

virtually any standard rack-mount hardware can serve as an MCBR without specialized silicon or proprietary firmware.

Arbitrary Traffic Steering. The coordinated routing design between OSS and ALB (§5.2) is an internal integration that every cloud provider can replicate within its own control plane. The only piece of cross-team coordination that this co-design demands is asking the ALB team to stash the original *vip* into a TCP option on rewrite (§5.2). Mainstream cloud ALBs are software-defined [9, 31], so a fixed-logic option-write hook adds neither a new packet-processing stage nor a new state machine. It is a localized patch to the existing rewrite path with no additional engineering barrier.

Inter-business QoS Isolation. The forwarding-plane customization between OSS and the underlying network fabric is likewise an intra-cloud integration. The more aggressive ingredient here is descending below IP to carve out dedicated OTU boards and DWDM wavelength blocks for a single business, a practice we are not aware of any other cloud having deployed at scale. The encouraging finding is that no new optical or networking primitive is needed: the entire isolation is realized by composing two long-standardized features, SRv6 and VRF.

7.2 Serving Other Applications with Lean

This paper focuses on OSS, but the Lean architecture is equally applicable to other cost-sensitive cloud services that generate heavy cross-AZ traffic, including Elastic Block Storage (EBS), message queues (MQ) and container image registries. OSS uses all three (R1/R2/R3) because it serves public clients, while internal-only businesses (EBS and MQ) typically need only R3.

The Lean controller is multi-tenant: each onboarded business is allocated a dedicated SID range, a quota of ESR uplink ports, and a DWDM wavelength block, all registered through the same control plane. These tenants are cloud services, not the $O(10K)$ enterprise customers of OSS. Lean keeps no state for individual customers, since a *SID* encodes an AZ or Edge-PoP, never a customer. Three classes of shared resources have ample headroom for many such tenants: the Lean controller stays below 0.5% CPU (Figure 14), ALB routing tables are at most 11.75% full (Table 5), and SRv6 *SID* consumption is under 0.11% (Table 6). The hard constraint is the number of physically independent ESR uplink ports (Figure 5). In practice, a production ESR exposes 256 equivalent ports, more than sufficient to isolate the handful of top-tier applications. EBS is already in production on Lean. Dedicating traffic resources does not remove failures of the shared equipment itself: when a chassis fails, every business on it is hit, and surviving that remains the job of each service's own redundancy, as with the 3AZ design of §2.2.

7.3 Optimizing Global Access Cost

While §3.2 focused on *ingress* access cost, the same arbitrary traffic steering capability (**R2**) also unlocks optimization of *egress* carrier cost at global scale. Public egress bandwidth is a significant expenditure, and each carrier contract includes a committed bandwidth floor: charges accrue at the committed level regardless of actual utilization. Under conventional BGP routing, the application layer has no influence over egress carrier selection. As customer coverage expands, new carriers must be procured to maintain reachability, but each introduces a fresh committed-bandwidth obligation. Table 8 shows the number of global carriers contracted per region. Even moderately sized regions require over a dozen carriers; the busiest areas exceed 35. If each carrier maintains an independent bandwidth pool with its own committed floor, the aggregate guaranteed minimum grows rapidly while per-pool utilization remains low. This creates a *compounding fixed-cost problem*: traffic is spread across an increasing number of contracts, utilization per carrier drops accordingly, yet committed fees are paid in full per contract.

Table 8. Number of different global carriers per area.

Area	#1	#2	#3	#4	#5	#6
#Global carriers	38	36	13	14	15	11

Our approach. The primary lever is not sourcing cheaper carriers but fully utilizing the bandwidth already under contract. Lean's arbitrary traffic steering lets the application layer directly control egress selection, steering traffic toward underutilized carriers without protocol-level constraints. Because existing contracted bandwidth is not yet saturated, the marginal cost of pulling additional traffic through Lean is zero, whereas signing a new committed contract incurs a fixed monthly expenditure, so improving utilization directly defers procurement. In production, this strategy yields a 14.9% reduction in global access cost.

7.4 Exploiting Capacity After Expansion

As discussed in §3.4, OSS already provides AZ-level fault tolerance through 3AZ replication, and therefore has a lower fault-redundancy requirement than shared infrastructure, allowing its isolated links to operate at a higher utilization target. However, the underlying network follows a capacity-doubling expansion model: when a link is upgraded, its capacity doubles instantaneously, causing the utilization to drop to half overnight (e.g., from 60% to 30%). This step-function capacity jump is an inherent property of the expansion model and is unaffected by business-level isolation.

OSS pursues aggressive cost efficiency and cannot afford to leave the newly added capacity idle. To compensate for this gap, Lean performs proactive traffic rebalancing immediately after an expansion event. Specifically, once the expansion completes, Lean leverages its built-in scheduling engine to migrate a portion of traffic from other AZs into the newly expanded AZ, filling the fresh capacity headroom. For example, the source AZ's utilization may decrease from 55% to 40%, while the expanded AZ's utilization recovers from 27.5% to

45%, achieving dynamic global rebalancing and preventing the new capacity from sitting idle over extended periods.

7.5 When the Network Fails

Over four years in production, Lean has handled over 100 access, over 15,000 cross-region, and over 100 intra-region network failures annually without severe SLO violations. We describe recovery in each tier below and a consistency incident during rollout in §7.6.

Access. MCBRs run in 1+1 pairs, so a single-device failure is transparent. The costlier case is losing a whole EdgePoP: DNS withdraws its virtual IPs and clients re-converge to the nearest remaining EdgePoP within seconds (§5.2).

Cross-region. Fiber cuts and device failures are routine, but the Dual-Plane Triple-Route backbone (§3.3), with BGP convergence and traffic replanning, absorbs them without affecting users.

Intra-region. The typical failure is an unreachable AZ. OSS's 3AZ redundancy already handles it (§2.2), and the 50% watermark lets the two surviving AZs carry the full load (§5.3).

7.6 Beware of Inconsistency Pitfall

OSS uses a Raft-based [29] consensus protocol to coordinate writes across AZs: the control plane determines which AZ replicas are available, and the data plane carries the actual object bytes. Isolating data-plane cross-AZ flows into a dedicated channel delivered significant cost savings, yet during the rollout we encountered a severe consistency incident.

When the isolated network path for a particular AZ experienced a failure, the Raft consensus nodes remained unaware of the outage and continued to consider the data nodes in that AZ writable. In reality, write operations on the data path were silently failing.

Through careful debugging we identified the root cause: when applying SRv6 *SID* tags to mark OSS traffic for isolation, only data-path flows had been tagged. The Raft control-path traffic was still carried over the shared network alongside ordinary traffic. As a result, the control plane could not sense data-plane network state: Raft saw all three AZs as healthy and continued dispatching writes, while the data plane to one AZ was completely unreachable.

After diagnosing the issue, we performed a thorough audit of all cross-AZ flows and applied the OSS *SID* tag uniformly to every data-plane and control-plane flow, eliminating the visibility gap between the two planes.

8 Related Work

Object storage systems. GFS [12], Azure Storage [8], Tectonic [30], and f4 [27] address storage-layer architecture, availability, and media cost reduction, treating the underlying network as a provisioned resource. Lean is complementary: it optimizes the *network* cost of object storage, which at our scale equals the storage cost in magnitude.

Cost-aware traffic steering. B4 [18, 23], SWAN [17], and BwE [26] optimize inter-DC WAN utilization through centralized TE. Pretium [24] adds cost awareness; Entact [33] selects among upstream carriers. At the edge, Espresso [37] and Edge Fabric [34] enable fine-grained peering selection. Each optimizes a single tier and treats applications as opaque demands, whereas Lean spans heterogeneous tiers with different cost structures and requires co-design with the application layer, jointly controlling ingress and egress paths while extending isolation into the optical transport layer.

Segment Routing and SRv6. Segment Routing [11] and its IPv6 instantiation [10] provide source-routed forwarding without per-flow state at transit nodes. Prior deployments use SR primarily for traffic engineering [16] or service chaining [1]. SRv6 itself is standard, like DNS steering. Lean's novelty is using it not as a TE primitive but as an application-infrastructure coordination primitive.

Network QoS and isolation. DiffServ [6] and IntServ [7] define per-class and per-flow QoS within a shared network. In the cloud context, Oktopus [3], SecondNet [15], and Silo [25] offer tenants bandwidth guarantees through virtual network abstractions. Network slicing [4] partitions physical resources into logically isolated slices for different service classes. All of these operate above the optical layer and rely on scheduling to enforce isolation. Lean's Inter-business QoS Isolation descends below IP into the optical transport, achieving *physical* rather than *logical* isolation.

Cloud network cost analysis. CloudCost [14] analyzes cloud network TCO; Skyplane [22] optimizes cross-cloud transfer cost via overlay paths. Lean contributes a complementary perspective: a structural cost analysis *within* a single cloud's object storage, revealing the counter-intuitive inverse relationship between distance and cost, and a deployed system that acts on it.

9 Conclusion

We presented Lean, a system that reduces OSS network cost through application-infrastructure co-design using SRv6 as a unifying primitive. Concretely, Lean instantiates three cross-product co-designs: OSS×EdgePoP (MCBR), OSS×ALB (arbitrary traffic steering), and OSS×OTN (inter-business QoS isolation). In production for 4 years across 7 regions and 75 EdgePoPs, Lean has cut the OSS network bill by 55.6%, saving hundreds of millions of dollars annually. The broader lesson is that layered abstraction lets a service scale fast, but coordinating across those layers lets the scaled service save.

Acknowledgments

We acknowledge all teams within Alibaba Cloud that contributed to the success of Lean, especially our colleagues on the OSS team and the infrastructure network team. We also thank our shepherd for guiding the final version of this paper, and the ATC reviewers for their insightful comments.

References

[1] Ahmed M. Abdelsalam, François Clad, Clarence Filsfils, Stefano Salsano, Giuseppe Siracusano, and Luca Veltri. Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[2] Amazon Web Services. Amazon S3 – Cloud Object Storage, 2025. https://aws.amazon.com/s3/.

[3] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 242–253, New York, NY, USA, 2011. Association for Computing Machinery.

[4] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Comput. Netw.*, 167(C), February 2020.

[5] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. Finding a needle in haystack: facebook's photo storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, page 47–60, USA, 2010. USENIX Association.

[6] David L. Black, Zheng Wang, Mark A. Carlson, Walter Weiss, Elwyn B. Davies, and Steven L. Blake. An Architecture for Differentiated Services. RFC 2475, December 1998.

[7] Robert T. Braden, David D. Clark, and Scott Shenker. Integrated services in the internet architecture: an overview. *RFC*, 1633:1–33, 1994.

[8] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 143–157, New York, NY, USA, 2011. Association for Computing Machinery.

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

[10] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. Segment Routing over IPv6 (SRv6) Network Programming. Internet-Draft draft-ietf-spring-srv6-network-programming-28, Internet Engineering Task Force, December 2020. Work in Progress.

[11] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. Segment Routing Architecture. RFC 8402, July 2018.

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, October 2003.

[13] Google Cloud. Cloud Storage – Online Data Storage, 2025. https://cloud.google.com/storage.

[14] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2009.

[15] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. *Proceedings of the 6th International COnference*, 2010.

[16] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. Blink: Fast connectivity recovery entirely in the data plane. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 161–176, Boston, MA, February 2019. USENIX Association.

[17] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):15–26, August 2013.

[18] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 74–87, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in windows azure storage. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, page 2, USA, 2012. USENIX Association.

[20] ITU-T. G.694.1: Spectral Grids for WDM Applications: DWDM Frequency Grid, 2020. https://www.itu.int/rec/T-REC-G.694.1.

[21] ITU-T. G.709: Interfaces for the Optical Transport Network, 2020. https://www.itu.int/rec/T-REC-G.709.

[22] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica. Skyplane: Optimizing transfer cost and throughput using Cloud-Aware overlays. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1375–1389, Boston, MA, April 2023. USENIX Association.

[23] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Suan Ong, Leonid B. Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: experience with a globally-deployed software defined wan. *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, 2013.

[24] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 73–86, New York, NY, USA, 2016. Association for Computing Machinery.

[25] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. Silo: Predictable message latency in the cloud. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 435–448, New York, NY, USA, 2015. Association for Computing Machinery.

[26] Alok Kumar, Amin Vahdat, Aspi Siganporia, Björn Carlin, C. Stephen Gunn, Enrique Cauich Zermeno, Jing Ai, Mathieu Robin, Mihai Amarandei-Stavila, Nikhil Kasinadhuni, Stephen Stuart, Sushant Jain, and Uday Naik. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *ACM SIGCOMM 2015*, 2015.

[27] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's warm BLOB storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, Broomfield, CO, October 2014. USENIX Association.

[28] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[29] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.

[30] Satadru Pan, Theano Stavrinos, Yunqiao Zhang, Atul Sikaria, Pavel Zakharov, Abhinav Sharma, Shiva Shankar P, Mike Shuey, Richard Wareing, Monika Gangapuram, Guanglei Cao, Christian Preseau, Pratap Singh, Kestutis Patiejunas, JR Tipton, Ethan Katz-Bassett, and Wyatt Lloyd. Facebook's tectonic filesystem: Efficiency from exascale. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 217–231. USENIX Association, February 2021.

[31] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. Ananta: cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 207–218, New York, NY, USA, 2013. Association for Computing Machinery.

[32] Yakov Rekhter, Tony Li, and Susan Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[33] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. Internet performance from facebook's edge. In *Proceedings of the Internet Measurement Conference*, IMC '19, page 179–194, New York, NY, USA, 2019. Association for Computing Machinery.

[34] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Ítalo F. S. Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017.

[35] Daniel Senie and Paul Ferguson. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[36] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, High-Performance distributed file system. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*, Seattle, WA, November 2006. USENIX Association.

[37] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 432–445, New York, NY, USA, 2017. Association for Computing Machinery.