Gryphon: Scaling Hyperscale Multi-Tenant Gateways Beyond the Petabit-Era via DPU-Augmented Hierarchical Co-Offloading

Yuemeng Xu[†], Haoran Chen[*], Jiarui Guo[†], Mingwei Cui[*], Qiuheng Yin[†], Cheng Dong[*], Peng He[*], Chenmin Sun[*], Yangyujia Wang[†], Daxiang Kang[*], Xian Wu[*], Yang Gao[*], Lirong Lai[*], Kai Wang[*], Zhuochen Fan[†], Tong Yang[†], Hongyu Wu[*]

[†]State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University [*]ByteDance

Abstract

At ByteDance, cloud gateway clusters orchestrate petabit-scale aggregate traffic. Traditional ASIC-only gateways fail to meet these escalating demands due to severe on-chip resource constraints and limited programmable flexibility, while pure software solutions or alternatives like disaggregated SmartNICs struggle to match terabit-scale line-rate throughput. To bridge this gap, we present Gryphon, a hyperscale cloud gateway built on a hybrid architecture that integrates DPUs directly into the switching ASIC's forwarding path. This design resolves the fundamental tension between capacity and speed, expanding table scale by up to 1000× and augmenting programmability, while sustaining 1.6 Tbps line-rate throughput at a cost of only ∼8 μs in additional average latency. To manage this hardware heterogeneity, we introduce Hierarchical Co-Offloading (HLCO) in the data plane, achieving >99.9% fast path hit rate, while retaining software fallback for complex operations. In the control plane, we develop an abstraction layer (P4Bridge) that decouples hardware specifics from policy configuration. Gryphon has been operating at production scale for over a year, deployed on hundreds of nodes across multiple Availability Zones. We also share production measurements and operational experiences that serve as the first hyperscale-proven guidelines for next-generation DPU-augmented cloud gateways.

CCS Concepts

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

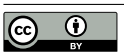
Keywords

cloud gateways, programmable data planes, DPUs, hierarchical offloading, multi-tenant cloud networks

ACM Reference Format:

Yuemeng Xu, Haoran Chen, Jiarui Guo, Mingwei Cui, Qiuheng Yin, Cheng Dong, Peng He, Chenmin Sun, Yangyujia Wang, Daxiang Kang, Xian Wu, Yang Gao, Lirong Lai, Kai Wang, Zhuochen Fan, Tong Yang, and Hongyu Wu. 2026. Gryphon: Scaling Hyperscale Multi-Tenant Gateways Beyond the Petabit-Era via DPU-Augmented Hierarchical Co-Offloading. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3789240.3829119

The first three authors contribute equally. Tong Yang (yangtong@pku.edu.cn) and Hongyu Wu (hongyu.wu@bytedance.com) are corresponding authors.



SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829119

1 Introduction

Managing petabit-scale aggregate traffic while maintaining strict requirements for tens of thousands of tenants presents unprecedented challenges in scale and complexity for centralized cloud gateways. Situated at the boundaries of datacenters, these gateways mediate a massive mix of intra-datacenter (east-west) and external (north-south) traffic [1, 2], maintaining extensive dataplane entries for workloads ranging from throughput-intensive geo-distributed data synchronization [3] to latency-sensitive interactive services [4]. To sustain this, a single gateway instance must deliver microsecond-level latency, terabit-scale throughput, and accommodate hyperscale forwarding tables while remaining cost-efficient at scale [5, 6].

However, as cloud workloads evolve, the explosive growth of forwarding and policy rules, coupled with the requirement for strict isolation, has pushed commodity hardware architectures to their limits. This strain manifests in three representative production workloads: ① Hyperscale tables (Doubao [7]): To strictly isolate thousands of tenants and services on our LLM platform, the control plane installs massive fine-grained forwarding rules and access-control lists (ACLs), quickly exhausting the limited on-chip memory (SRAM/TCAM) of commodity programmable switches [8]. ② Low latency (TikTok [4]): Interactive live streams require real-time server-side processing within cloud compute pools. Video flows are ingested at the edge and routed to the cloud core, demanding tight jitter and tail-latency bounds under contention [9]. ③ High throughput (Volcengine [3]): Geo-redundant storage replication generates long-lived elephant flows that stress the gateway dataplane and can saturate its links, requiring high-throughput dataplane processing.

These diverse and demanding workloads expose the inherent limitations of existing gateway paradigms: Software gateways (CPU-based) offer high flexibility but fail to scale with petabit-scale aggregate traffic due to packet processing bottlenecks [10]. Conversely, while programmable switching ASICs (e.g., Tofino) deliver terabit-scale line-rate forwarding, they are constrained by severe on-chip memory limits, which remain insufficient to accommodate hyperscale forwarding tables even with algorithmic optimizations [6]. Alternatives such as FPGA-based gateways, while providing greater flexibility than ASICs, incur substantial development complexity and significantly higher cost [5]. This landscape reveals a critical gap: no existing architectures can simultaneously deliver the hyperscale capacity, deterministic performance, and rapid feature evolution required by modern cloud networks.

A conventional approach to managing escalating workloads is horizontal scaling (scale-out) — multiplying processing capability by clustering additional nodes. However, this strategy has become unsustainable for hyperscale gateways. For software-based clusters, a limited per-core packet-processing budget makes the system highly susceptible to hot-core (CPU hotspot) saturation under heavy-hitter–induced skew and microbursts [11–13]. Such single-core saturation inevitably compromises performance isolation among multiple tenants, leading to severe Service Level Agreement (SLA) violations [14]. For ASIC-based solutions, scaling out to overcome memory limits necessitates fragmenting forwarding tables across multiple nodes, introducing immense complexity in inter-device traffic steering and state synchronization. Consequently, there is an urgent need for a scale-up architecture that enhances per-node capacity to sustain the demand of hyperscale workloads.

Integrating P4-programmable DPUs with established programmable switching ASICs offers a promising path to overcome these limitations. Modern DPUs feature massive DDR4/5 DRAM that breaks the on-chip SRAM capacity ceiling, and programmable MPUs capable of executing sophisticated logic beyond the fixed-stage constraints of ASIC pipelines. This synergy creates a robust scale-up instance that simultaneously delivers hyperscale state capacity and terabit-scale line-rate throughput, all while incurring a latency overhead on the order of microseconds — a trade-off fully acceptable for strict production SLAs [15].

Realizing this heterogeneous architecture, however, is non-trivial and requires addressing three fundamental challenges:

- **Bridging the gap between host-centric DPUs and network-centric cloud gateways.** DPUs are primarily designed for per-host offloads, and typically provide 100–400 Gbps per device [16]. This is far below the Tbps-scale per device throughput demanded by hyperscale cloud gateways. Repurposing these endpoint devices for the network border introduces a fundamental performance mismatch [5].
- **Orchestrating traffic across asymmetric hardware resources with extreme performance.** Switching ASICs deliver deterministic sub-microsecond match–action pipeline latency but are limited by scarce on-chip resources (SRAM/TCAM) [8, 17]. In contrast, DPUs provide GB-scale forwarding state in off-chip DDR memory, but incur microsecond-scale latency [18] and low sustained throughput [15]. Co-offloading traffic to harness these complementary strengths while mitigating their weaknesses is a significant challenge.
- **Shielding the control plane from hybrid datapath complexity.** Exposing raw hardware heterogeneity to upper layers impedes development velocity and hinders cross-team collaboration. The challenge lies in designing a unified abstraction that enables operators to define logical policies agnostic to the underlying hardware, automatically mapping them to the appropriate target.

To address these challenges, we present Gryphon, the first heterogeneous, hyperscale, multi-tenant cloud gateway that integrates programmable switching ASICs (Intel Tofino [19]) with P4-programmable DPUs (AMD Pensando [20]) into a single logical forwarding pipeline. We physically integrate four P4-programmable Pensando DPUs [20] with a folded Intel Tofino pipeline [21], aligning per-device bandwidth with gateway-scale throughput and achieving 1.6 Tbps line-rate forwarding. Centrally, we introduce Hierarchical Co-Offloading (HLCO), a tri-tier datapath that extends the classic fast/slow-path model [22]: the switching ASIC executes high-throughput, low-complexity processing, the DPU forms a flexible middle tier to host large tables and richer functions, and software provides a fully programmable fallback for exceptional or complex tasks. Finally, we design P4Bridge, a unified abstraction that exposes the ASIC-DPU datapath as a single logical pipeline, enabling transparent rule configuration across heterogeneous hardware. Our major contributions are summarized as follows:

- We present Gryphon, the first cloud gateway that integrates DPUs and switching ASICs into a unified programmable P4 pipeline. It achieves 1.6 Tbps line-rate forwarding and leverages DPU memory to expand table capacity by 10–1000×, effectively eliminating resource hotspots on switching ASICs.
- We propose HLCO to efficiently partition tasks across the heterogeneous data plane. It offloads over 99.9% of traffic to the hardware fast path, effectively eliminating software bottlenecks while maintaining software flexibility.
- We design P4Bridge, a hardware-agnostic framework that abstracts the hybrid ASIC-DPU datapath into a single logical pipeline. We also introduce optimization techniques, including compact metadata encoding and table coalescing, to maximize interconnect efficiency and reduce memory access latency.
- We share experiences and lessons from a large-scale production deployment spanning over one year. We analyze critical design trade-offs, providing the first hyperscale-proven guidelines for designing next-generation DPU-augmented cloud gateways.

Ethics: This work does not raise any ethical issue.

2 Background and Motivation

2.1 Anatomy of Cloud Gateways

Roles of cloud gateway. In public clouds, providers multiplex a shared physical infrastructure into tenant-isolated virtual networks (e.g., VPCs). Each tenant operates within a private network abstraction with independent addressing, routing, and ACLs, requiring the underlying network to preserve strong isolation and fulfill SLA requirements [6, 8]. As illustrated in Figure 1 and Table 1, the cloud gateway (CGW) serves as the central hub at critical traffic aggregation points: it orchestrates various connectivity patterns, connecting in-cloud VMs through the vSwitches, peering with remote CGWs and on-premises datacenters (IDC) over the cross-region network (CRN) [6]. Operating at this junction, CGWs must sustain hyperscale throughput while enforcing fine-grained, multi-tenant forwarding policies.

Characteristics of hyperscale multi-tenant cloud gateway. ① A significant portion of traffic is *highly latency-sensitive* [23, 24]. Critical applications, such as LLM inference, real-time collaboration, impose strict latency limits, where even minor delays can degrade service quality and compromise service level objectives (SLOs) [25]. ② Traffic is characterized by its *immense aggregate volume*, reaching the petabit-per-second (Pbps) level as a direct consequence of simultaneously serving numerous bandwidth-hungry

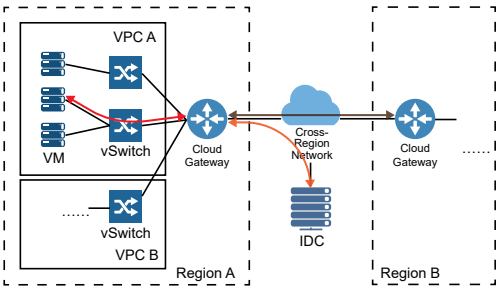


Figure 1: Cloud gateway as the central hub of traffic.

workloads [1, 4]. ③ The gateway must maintain *massive per-tenant states*, including VPC routing, ACLs and VM-NC mappings, to enforce tenant isolation. This results in an inevitable explosion of forwarding tables at the hyperscale traffic hub [6, 21]. ④ Traffic exhibits a *highly skewed flow size distribution* [26], characterized by long-lived, high-volume elephant flows alongside a vast number of short-lived mice flows.

Design goals of hyperscale multi-tenant cloud gateway: ① **Stringent performance.** The gateway must deliver microsecond-level latency and terabit-scale throughput to satisfy SLAs across diverse, bandwidth-intensive cloud workloads [1, 2, 21]. ② **Hyperscale table capacity.** The gateway must maintain massive routing and policy tables to ensure tenant isolation [5]. ③ **Cost efficiency.** At petabit scale, minimizing Total Cost of Ownership (TCO) is important, which requires co-optimizing hardware procurement (CapEx) and power consumption (OpEx) through judicious architectural choices. ④ **Flexibility.** The gateway must support rapid, safe updates to forwarding policies and packet-processing functions to enable fine-grained programmability and incremental feature rollouts [5].

2.2 Evolution towards Gryphon

The first generation: Software-only cloud gateways. Our initial cloud gateway adopted a fully software-based architecture [27] and parallelized packet processing [28, 29]. Driven by the need for rapid feature evolution, this design leveraged mature user-space packet I/O ecosystems (e.g., DPDK [30], Netmap [31]) to accelerate development cycles.

However, as traffic grew, tenant isolation became critical. While fixed per-attachment rate caps (e.g., a few Gbps in our deployment) helped contain noisy neighbors, it became ill-suited for high-volume east-west traffic [6], leading to unnecessary queuing and packet loss under bursts. More fundamentally, scalability was often limited by traffic skew: hash-based mechanisms, such as Equal-Cost Multi-Path (ECMP) and Receive-Side Scaling (RSS), pinned long-lived elephant flows to specific paths or cores, and hash collisions among these flows created persistent hotspots [32, 33]. As a result, aggregate throughput was frequently bottlenecked by the most congested core, reducing the effectiveness of horizontal scaling under skewed workloads [32].

The second generation: Tofino-based gateways. To overcome software bottlenecks, we transitioned to Tofino-based gateways, which deliver 1.6 Tbps line-rate throughput while achieving microsecond-level forwarding latency.

Table 1: Connectivity provided by the cloud gateway.

Connectivity goal	Connectivity chain
internal VMs	CGW ↔ vSwitch ↔ VM
remote CGWs (inter-region)	CGW ↔ CRN ↔ CGW
IDC (on-premises)	CGW ↔ CRN ↔ IDC

However, the Tofino ASIC suffered from critically scarce on-chip memory [5], making it challenging to support large-scale table entries, particularly the VM-NC tables mapping virtual machines (VMs) to their corresponding physical server addresses. Even with pipeline-aware resource packing and IPv4/IPv6 key compression, SRAM utilization frequently approached saturation. Table 2 illustrates this bottleneck, showing SRAM depletion across twelve consecutive stages within a critical pipeline where dependencies forced the placement of massive VM-NC tables. Furthermore, the rigid match-action pipeline lacked specialized engines for computationally intensive tasks — such as advanced load balancing, flow logging, and encryption — and could not support the elastic buffering required for stateful operations [34–36]. Consequently, these tasks were frequently punted back to software, resulting in unpredictable performance and increased operational complexity.

Towards the third generation: Hybrid programmable gateway based on heterogeneous hardware. Prior generations reveal a fundamental gap between software flexibility and ASIC performance, necessitating a new hardware tier to bridge them. This tier must operate in tandem with Tofino, and deliver high programmability, competitive performance, and sufficient memory for hyperscale tables. We evaluated FPGAs and DPUs as candidates, and selected DPUs for their practical advantages in large-scale cloud deployments:

- **Higher efficiency and lower TCO.** Our benchmarking conducted under identical 2U chassis (same form factor and 1.6 Tbps line-rate forwarding) shows that DPU-based gateways reduce power consumption by 21% and hardware cost by 14% compared to commercial FPGA-based designs [37, 38]. This significantly lowers Total Cost of Ownership (TCO) at scale.
- **Reduced development complexity.** DPUs leverage a standard Linux environment and mature networking frameworks (e.g., C/C++, P4, and DPDK), enabling rapid iteration [30, 39]. In contrast, FPGA-based designs typically require development at the register-transfer level (RTL) or through high-level synthesis (HLS). They also involve longer synthesis and verification cycles, which significantly increases engineering overhead [40–43].
- **Architectural compatibility.** DPUs and P4 switches share a similar match-action abstraction. While the ASIC provides deterministic, TCAM-assisted prefix forwarding, the DPU's large DRAM and general-purpose compute support massive exact-match states and complex logic. This alignment facilitates a natural hierarchical design and simplifies functional migration between components [44].

2.3 P4-programmable DPU

To address the limitations of switching ASICs, we integrate P4-programmable DPUs into our next-generation gateway design. As shown in Figure 2, the DPU provides a tightly coupled system-on-chip design, combining a programmable P4 dataplane with a

Table 2: SRAM Utilization Hotspot in the Most Critical Pipeline Stages.

Metric	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	Average
SRAM Util.	**100.0%**	**100.0%**	**100.0%**	97.5%	98.8%	98.8%	98.8%	**100.0%**	98.8%	**100.0%**	98.8%	77.5%	**97.4%**

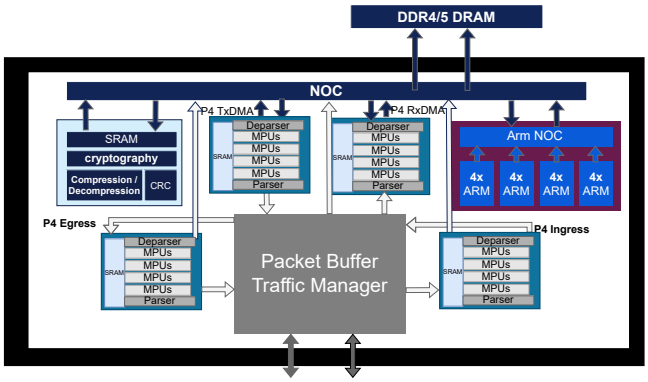


Figure 2: Simplified architecture of the DPU design.

general-purpose processing complex (ARM cores) that runs the DPDK-based datapath runtime.

The P4-programmed ingress and egress pipelines implement performance-critical dataplane functions such as access control (ACL), metering, routing, encapsulation/decapsulation, and lightweight NAT. To sustain large tables, the DPU employs on-chip SRAM as low-latency cache and off-chip DDR4/5 DRAM for high-capacity state storage.

Paired with dedicated Tx/Rx DMA engines, the DPU can achieve zero-copy data movement between P4 pipelines and system memory. This hybrid design bridges ASIC-class performance with software-defined flexibility, enabling scalable, low-latency packet processing and eliminating the need for host-side CPUs by running DPDK [30] runtime directly on embedded ARM cores.

3 Gryphon's Architecture

3.1 High-Level Architecture

We extend the traditional fast/slow path model to a three-tier hierarchy: (1) switching ASICs as the ultra-fast path, (2) inline P4 pipelines on DPUs for mid-tier offload, and (3) ARM cores on DPUs handling the ultimate fallback. This hierarchical offloading model preserves performance for the common case while retaining software flexibility.

Figures 3(a) and 3(b) summarize our hierarchically co-offloading design. At a high level, the end-to-end packet-processing pipeline follows a three-stage datapath: a pre-DPU Tofino pipeline, a DPU pipeline, and a post-DPU Tofino pipeline (Fig. 3(a)). Upon pipeline entry, each packet is tagged with a Service Identifier (SvcID) for classification and subsequent processing. All packets traverse the two Tofino stages, while the traffic manager steers only the packets that need additional state or computation to the DPU; the rest bypass the DPU stage and directly resume in the post-DPU pipeline.

The DPU stage (Fig. 3(b)) integrates a programmable pipeline, ARM cores, and hierarchical memory (SRAM/DRAM). The pipeline performs exact-match lookups on DRAM-backed tables, utilizing on-chip SRAM as a low-latency cache. The pipeline accesses on-chip SRAM with deterministic latency, whereas external DRAM is managed via asynchronous lookup/DMA engines.

Upon a lookup hit (whether in SRAM or DRAM), the DPU returns the result to the post-DPU Tofino pipeline to complete the forwarding actions. Conversely, a lookup miss triggers an exception to the ARM cores for slow-path processing. The ARM cores resolve the forwarding decision and install the corresponding flow entries, ensuring subsequent packets are handled entirely in the hardware fast path.

Path 1: Switch-only path. This path handles traffic not destined for local VMs (e.g., cross-region, Internet, or IDC-bound). After traversing the pre-DPU Tofino pipeline, the packet is steered to bypass the DPU stage. It then proceeds directly to the post-DPU pipeline for forwarding, preserving the low-latency line-rate performance of the switching ASIC (Fig. 3(a)).

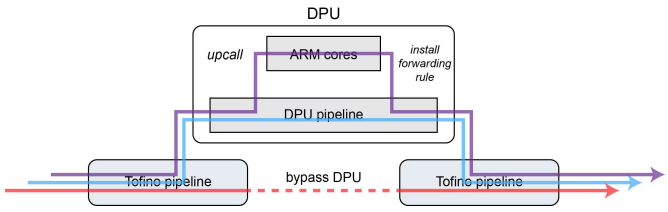
Path 2: Hybrid fast path. For traffic destined to local VMs, processing requires resolving the VM-NC table to identify the target physical host and determining the intra-datacenter next hop. However, the limited on-chip memory and processing stages of the Tofino ASIC are insufficient to accommodate these massive table entries. Consequently, such packets are forwarded to the DPU for exact-match lookups in SRAM/DRAM. Lookup hits return to the post-DPU pipeline where subsequent forwarding actions are executed (Fig. 3(a)); conversely, lookup misses are punted to the ARM cores.

Path 3: Software slow path. When the DPU pipeline fails to resolve a flow (i.e., a lookup miss)—typically triggered by the arrival of the first packet of a new flow—the packet is redirected to the ARM cores for software processing. The software agent executes the control logic and installs corresponding forwarding entries, effectively offloading future traffic of the same flow to the Hybrid Fast Path (Path 2), bypassing the software stack.

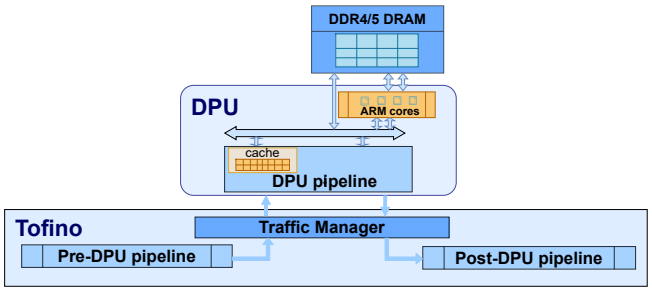
3.2 Hardware Organization

Standard Tofino architectures operate four pipelines in parallel to maximize throughput, where each packet is processed by the ingress and egress logic within the same pipeline [35]. However, for our logic-heavy cloud gateway, the primary bottleneck is not throughput, but the scarcity of processing stages and memory resources required to execute complex forwarding chains [21]. Deployment data indicates that the stage and memory requirements of the most resource-intensive tables alone are sufficient to exhaust the resources of a standard pipeline. Furthermore, the integration of other functions (e.g., metering, ACL) introduces significant additional logic and memory footprints.

Figure 4 shows the detailed pipeline organization. We configure the internal ports between pipelines (pipes in Fig. 4) in a mode that chains the four pipelines sequentially. As illustrated, packets enter the first pipeline (pipe 3), traverse the ingress logic, and are redirected via the traffic manager to the egress logic of the subsequent pipeline (pipe 1), continuing this process until they reach the egress logic of the final pipeline (pipe 3) and exit the system. This "pipeline folding" to chain four physical pipelines into a single processing pipeline effectively multiplies the available Tofino resources by



(a) Logic overview of three typical datapaths for different types of traffic.



(b) Component view of the Tofino–DPU dataplane.

Figure 3: Architecture overview of hierarchically co-offloading design.

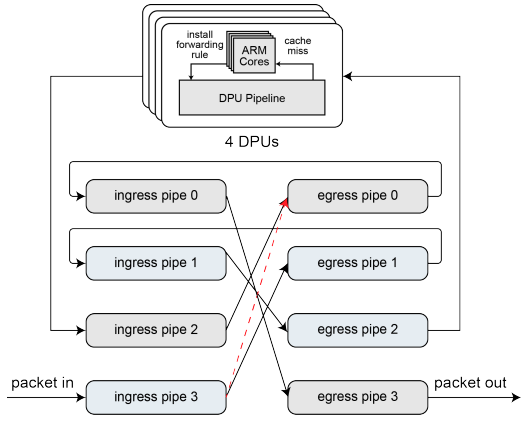


Figure 4: Datapath organization with folded pipeline and sandwiched DPUs.

4× to support rich forwarding logic, but at the cost of reducing aggregate throughput from 6.4 Tbps to 1.6 Tbps.

To match this line rate, we provision four DPUs in parallel and use ECMP to distribute packets to the DPUs, with packets entering and exiting the DPU via the same Tofino pipeline (pipe 2). This "sandwich" design co-locates the DPUs between the ingress and egress stages of the same Tofino pipeline (pipe 2), ensuring full-duplex bandwidth without sacrificing external port capacity.

4 Heterogeneous Pipeline

For a hyperscale cloud vendor, the datapath must support complex forwarding logic while adhering to strict hardware constraints. Instead of enumerating every corner case, we present a representative packet processing pipeline that illustrates the lifecycle of a typical VXLAN-encapsulated tenant packet across the heterogeneous Gryphon's architecture.

Hardware-guided task partition. Our design strategically maps different table types and processing logic across heterogeneous hardware based on their respective strengths. Leveraging the DPU's abundant SRAM and DRAM to overcome Tofino's on-chip memory scarcity, we offload hyperscale exact-match tables — most notably the VM-NC table — to the DPU. Conversely, the Tofino ASIC retains LPM lookups, utilizing its on-chip TCAM for deterministic prefix-based forwarding. Moreover, complex functions such as encryption, fine-grained flow logging, and ZooRoute [45] are executed on the DPU to leverage its specialized hardware accelerators and more flexible programmable pipelines. Finally, for lookup misses or policies requiring complex multi-step decisions, the DPU falls back to the software slow path, where the DPDK runtime (running on ARM cores) executes complex processing.

For protocol and control packets, DPU involvement is unnecessary. Allowing them to traverse the whole pipeline not only lengthens the processing path but also introduces potential risks to stability. To this end, we divert such traffic at ingress pipe 3 in Figure 4, bypassing the DPU and delivering them directly to the CPU for control-plane processing as shown by the red path in Figure 4, ensuring that control-plane traffic follows the shortest and most stable path.

Pre-DPU Tofino processing (ingress3 → egress1 → ingress1 → egress2 → DPU, in Figure 4). Pre-DPU processing follows three stages (detailed in Figure 5): (1) Stage 1: Traffic classification. Packets first hit a classification table that separates (a) VXLAN-encapsulated tenant traffic, identified by outer headers (e.g., UDP/VXLAN and remote VTEP IP), from (b) protocol/control traffic. Only VXLAN traffic proceeds through the subsequent stages. (2) Stage 2: Tenant/context resolution. Next, a tenant identification table maps the VNI to a tenant context ID, which selects the corresponding (logically isolated) policy-routing namespace. This indirection is crucial for isolation: each tenant is bound to its own policy-routing instance/partition. (3) Stage 3: Routing and forwarding. Within a tenant, multiple routing instances (e.g., subnets or security domains) may coexist; we select the routing instance using tenant context and a domain tag (e.g., derived from the inner source IP/subnet that associates source prefixes with routing domains or security contexts). Finally, a nexthop table materializes the forwarding action, including the output port and encapsulation parameters. Notably, the next hop applies only to outbound traffic not destined for local VMs, as the inbound next hop is resolved by the DPU.

DPU Processing. The DPU processes traffic destined for local VMs by leveraging its large table/memory capacity and flexible programmable logic. As illustrated in Figure 6, the DPU employs a hybrid architecture that integrates a P4-programmable hardware fast path with a DPDK-based software slow path. Upon entering the DPU's P4 ingress pipeline, packets undergo parsing and a flow table lookup to retrieve the nexthop index, the version number, and the destination physical host (NC) address resolved via the VM-NC mapping. In the event of a lookup miss or version validation failure, the packet is redirected to the software slow path. Within this path, the DPDK program handles lookup misses, expired rules, and complex corner cases. The DPDK runtime performs tasks including enforcing software-defined ACLs and resolving VM–NC

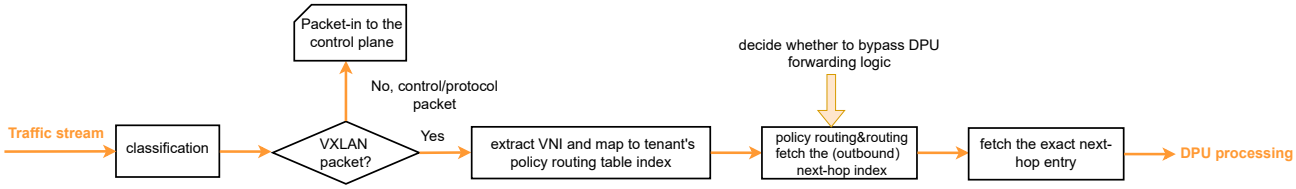


Figure 5: Table lookup sequence in the pre-DPU Tofino pipeline.

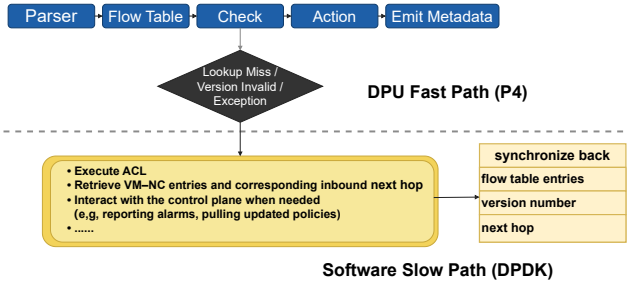


Figure 6: DPU packet processing pipeline with fast/slow path interaction.

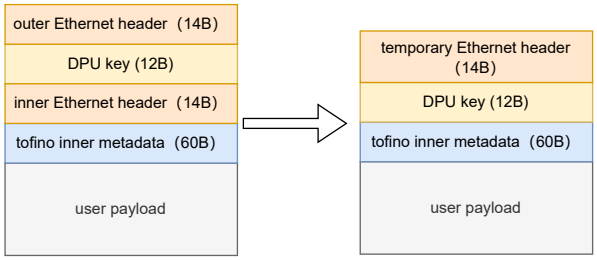


Figure 7: Reducing extra overhead on the Tofino–DPU interconnect.

mappings to determine the associated inbound nexthops. Additionally, it interacts with the control plane when needed to perform operations such as reporting runtime alarms. Once resolved, critical forwarding entries are synchronized back to the fast path, ensuring that subsequent packets from the same flow are handled entirely in hardware.

Post-DPU Tofino processing (DPU → ingress2 → egress0 → ingress0 → egress3 in Figure 4). After DPU processing, packets re-enter Tofino for a lightweight post-processing pass. Post-DPU Tofino pipeline typically (1) meters and enforces traffic budgets (e.g., per tenant), marking or dropping packets that exceed the configured rate; (2) handles packets that do not use VXLAN encapsulation by performing underlay Internet Protocol forwarding; (3) encapsulates overlay packets and attaches the selected underlay next-hop parameters; (4) recomputes and updates checksums after header rewrite or encapsulation to preserve packet correctness.

Metadata design optimization. Inter-hardware communication inevitably introduces header overhead, which reduces effective throughput because port bandwidth is bounded by the actual bytes transmitted on the link (especially under small packets) [21]. To minimize this penalty, we treat the Tofino–DPU interconnect as a transient internal link and apply two optimization techniques.

First, we employ compact handle encoding: Tofino pre-resolves large-table references and exports only a compressed 1–3 B index to the DPU. This yields a minimal 12 B *DPU key* (Figure 7, left).

Second, we implement header compression. In the uncompressed format, the outer Ethernet header carries Tofino–DPU interconnect metadata, while the inner one preserves the tenant header. Rather than stacking both internal and external Ethernet headers, we temporarily rewrite packets into a lean transit format for the interconnect and restore the original headers upon return (Figure 7, right). This saves 14 B per packet.

Combined, these techniques limit the heterogeneous traversal overhead to just 12 B, which translates into an effective-throughput gain on the order of 10 Gb/s in our deployment.

5 Control Plane Synergy

5.1 Hardware Abstraction

The vast architectural diversity among DPU vendors leads to distinct hardware architectures, programming abstractions, and resource constraints (e.g, varying capacities of TCAM, SRAM, or DRAM) [16]. Moreover, although both Tofino ASIC and Pensando DPU adopt P4 as a common programming language, their underlying microarchitectural differences limit the portability of implementation expertise across platforms [16, 19]. Control-plane developers, tasked with orchestrating control-plane policies (e.g., ACLs, forwarding rules), often operate distinctly from the hardware engineering developers, necessitating a clear abstraction interface, as requiring control-plane operators to manage low-level data plane nuances (across diverse targets like ASICs and DPUs) would significantly increase complexity and impede development velocity. To address this fragmentation, we design an abstraction interface (P4Bridge) to decouple control plane logic from hardware-specific constraints and variations in data plane implementations.

P4Bridge presents the gateway datapath as a single logical pipeline. Each logical table can carry a wide key beyond hardware constraints, as shown in Figure 8, and P4Bridge maps each logical table onto a set of hardware-resident physical tables, placing the corresponding match keys and actions into the appropriate stages and installing the concrete physical entries on the target hardware. Crucially, P4Bridge must preserve logical-table semantics despite underlying table splitting, coalescing, and remapping across ASICs and DPUs. P4Bridge also exposes consistent configuration options for header handling (e.g., DSCP/TTL) and a unified interface for counters and statistics. By hiding hardware-specific details (e.g., parser differences and how tables are laid out or looked up), P4Bridge lets control-plane developers program against a stable API while allowing hardware engineers to optimize each backend transparently.

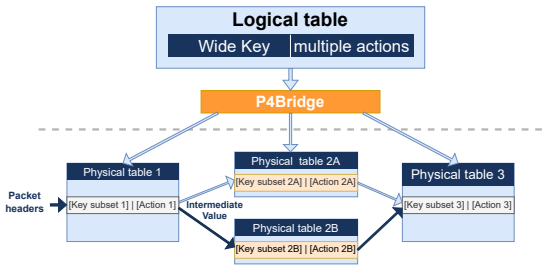


Figure 8: An example of table mapping via P4Bridge.

5.2 Table Coalescing

DPUs offer very limited TCAM, making hardware-efficient longest-prefix matching (LPM) impractical; even the Tofino ASIC's TCAM is not large enough to support all LPM operations [21, 46]. Instead, when implementing prefix-based policies or routing rules using DPUs and Tofino SRAM, prefix-based matching is typically realized via hash-indexed exact-match lookups in SRAM/DRAM. A direct LPM-to-hash realization requires probing multiple candidate prefixes sequentially, and each probe may further consult an overflow structure to resolve collisions—multiplying memory accesses and control decisions [47]. This overhead is directly manifested in the DPU datapath as additional pipeline stages and control logic, increasing per-packet processing cost and pressure. To mitigate this, we coalesce hash tables to reduce the number of probes and per-packet dispatch overhead.

Our key observation is that, in cloud gateways, prefix overlaps are rare in practice. We define two prefixes as overlapping when one prefix covers another more-specific prefix. Prefixes that are non-overlapping under this definition cannot produce competing LPM candidates, allowing P4Bridge to coalesce their otherwise dependent hash lookups into a single table. In our production configurations, such overlap is low, affecting only about 9.33% of IPv4 prefixes and 6.63% of IPv6 prefixes.

Exploiting this property, the control plane performs conflict detection before installing rules: if a set of prefixes can be guaranteed non-overlapping under our matching semantics, we consolidate their entries into a single hash table (and a shared overflow structure if needed). We further provide configuration guidelines that encourage non-overlapping policies and maximize coalescing opportunities. In the common case without multiple matches, coalescing substantially reduces both lookup complexity and memory footprint on DPUs.

5.3 Version-based Update

Table updates are particularly challenging in heterogeneous hardware environments. Forwarding rules span the Tofino pipeline, the DPU pipeline, and off-chip memory, each exposing different state representations and update primitives (e.g., fast-path tables updated via table-management interfaces vs. software-resident state updated through general-purpose memory operations under software-defined consistency) [44, 48]. Synchronizing these views consistently is non-trivial, especially given the mismatch in update timescales.

To this end, we keep mutable state in DRAM and run synchronization on the DPU's ARM cores, leveraging standard control-plane concurrency primitives. We compute a per-tenant service ID (SvcID) at the beginning of the Tofino pipeline and carry it as metadata to the DPU pipeline. Each service object is also tagged with a version number that reflects the latest configuration. The fast path only performs a lightweight version validation; upon detecting an expired version, it redirects the packet to the slow path for recomputation (Path 3 in Fig. 3(a)). After recomputation completes, the slow path installs the updated table entries into the fast path with the latest version. Upon tenants' reconfiguration, the controller increments the version in the Tofino-side table. Version-based update eliminates the need for tightly synchronized updates across components, while adding only a single version comparison on the fast path.

Table 3: Production traffic distribution across Gryphon's hardware fast paths.

Path	Relative share
Switch-only path	~17%
Hybrid fast path	~83%

6 Deployment

Gryphon has been deployed in production as part of ByteDance's cloud gateway infrastructure [3], supporting both north–south and east–west traffic. It is currently deployed across hundreds of gateway nodes within multiple clusters in five availability zones (AZs), with each node featuring a folded Tofino ASIC coupled with four Pensando DPUs.

The system has operated continuously in production for over one year, handling a significant portion of production traffic and sustaining multi-terabit-per-second aggregate throughput under high-concurrency workloads. The deployment continues to expand, gradually replacing previous-generation gateways that cannot meet current performance and scalability demands, with its footprint continuously growing as it integrates into an increasing number of cloud services and workloads.

7 Implementation

We implemented Gryphon on an in-house gateway platform [49]. The system is equipped with an Intel Tofino 6.4 Tbps ASIC featuring 64 × 100 Gbps ports. We integrated four AMD Pensando Elba DPUs [20], each featuring a 16-core ARM Cortex-A72 SoC for localized, complex processing. To bridge the DPU and the switching ASIC, we utilized breakout Active Optical Cables (AOCs) to split each 200 Gbps DPU port into two 100 Gbps links. This "sandwich" configuration ensures full-duplex bandwidth and aligns the DPU interconnect with the gateway's 1.6 Tbps line-rate throughput requirements.

8 Evaluation

8.1 Real-world Traffic Characterization

We present real-world traffic measurements collected from our production deployment to demonstrate the scale and operational characteristics of Gryphon.

Year-long production traffic. Figure 9 illustrates the aggregate traffic throughput over a one-year period. The data reveals a steady upward trend, peaking at over 1 Pbps. The sheer magnitude of this

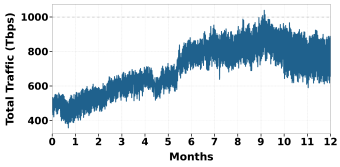


Figure 9: Year-long production traffic.

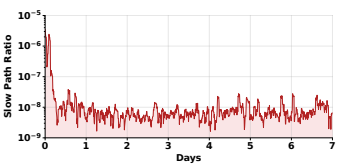


Figure 10: Slow path traffic ratio.

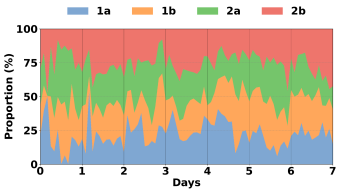


Figure 11: Intra-DPU traffic distribution.

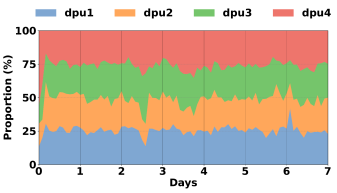
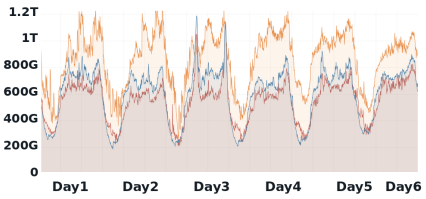
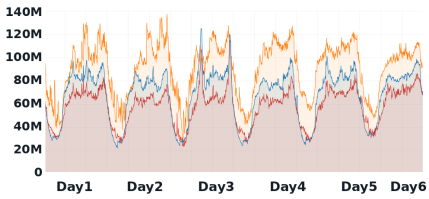


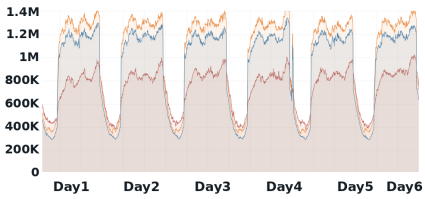
Figure 12: Inter-DPU traffic distribution.



(a) Throughput.

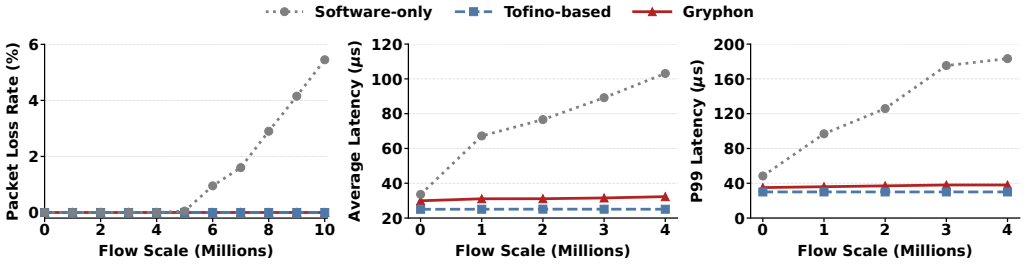


(b) Packet rate.



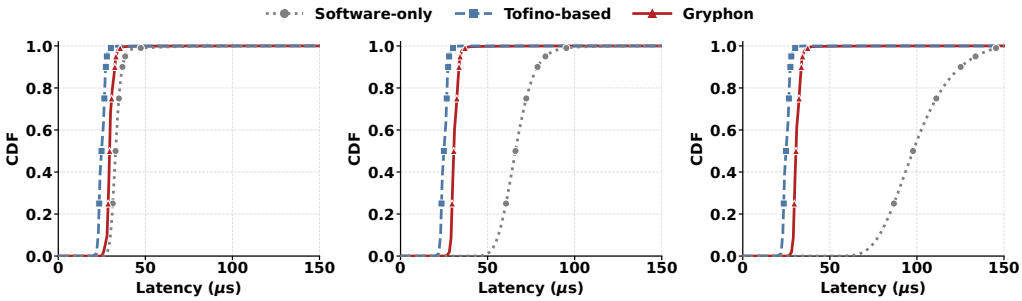
(c) Active flows.

Figure 13: Workload statistics from one representative production cluster.



(a) Packet Loss Rate (b) Average Latency (c) P99 Latency

Figure 14: Performance comparison of three cloud gateway versions



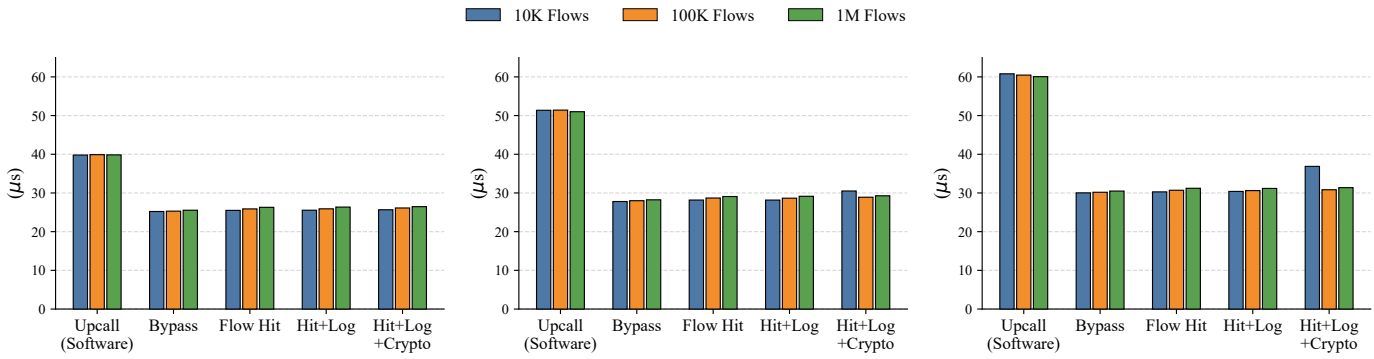
(a) 1K (b) 1M (c) 4M

Figure 15: Latency distribution with different concurrent flow scales

traffic underscores the critical need for a gateway architecture to deliver massive throughput beyond the capabilities of traditional software-only solutions.

Six-day production workload. Figure 13 presents production workload traces collected over a six-day period. The traces exhibit clear diurnal patterns while remaining at consistently high load: cluster throughput reaches the Tbps scale, packet rate reaches hundreds of Mpps, and flow pressure remains persistently high. This confirms that production gateways must simultaneously handle high bandwidth, high packet rate, and large forwarding-state pressure, rather than optimizing for only one dimension.

Production path distribution. To better understand Gryphon's production behavior, we analyze the traffic distribution across the three paths defined in Section 3.1: the Switch-only path, the Hybrid fast path, and the Software slow path. We find that slow-path traffic is mainly triggered by transient events such as bursts of new-flow arrivals, after which it quickly returns to a low steady level. As illustrated in Figure 10, the Software slow path consistently accounts for a very small fraction of traffic, staying below 0.001% for the vast majority of the time. This observation is consistent with our hierarchical co-offloading design, which keeps the common cases in hardware and punts only rare cases to software. Moreover, the Software slow path in our current deployment processes packet headers



(a) Average latency. (b) P99 latency. (c) P99.9 latency.

Figure 16: DPU forwarding latency under different pipeline configurations and flow scales. *Flow Hit* is the hardware fast-path baseline; *Hit+Log* additionally writes per-flow counters; *Hit+Log+Crypto* further enables inline encryption/decryption.

only, without inspecting payloads, which bounds per-packet work and reduces bandwidth and compute contention, making it less likely to become a bottleneck even when overall traffic is high.

We further classify a production traffic snapshot across the two hardware fast paths. As shown in Table 3, the resulting split is approximately 17:83: only about 17% of traffic follows the Switch-only path, whereas the remaining 83% uses the Hybrid fast path and therefore benefits from DPU-resident state capacity or inline processing. This workload composition shows that pure ASIC forwarding is important but not dominant in production cloud gateways; most traffic still relies on the DPU tier for large tables, or advanced gateway functions.

DPU load balance. As illustrated in Figure 11, intra-DPU traffic distribution is susceptible to skew in our production environment. Such imbalances are often inevitable due to our per-flow pinning policy, which ensures that all packets in a flow follow an identical path to prevent reordering. To mitigate this, we employ a dual-mode steering strategy that reactively rebalances traffic across the gateway complex. Figure 12 demonstrates the effectiveness of this approach: the stacked area chart shows that the load is evenly distributed across all four DPUs over a one-week period, with each DPU consistently handling approximately 25% of the aggregate traffic (additional data for other nodes are provided in Appendix B).

8.2 Performance

To evaluate Gryphon's operational boundaries, we conduct tests in a real production environment. By sweeping the number of active flow entries from 1K to 10M, we measure packet loss rate and end-to-end latency to assess how each design manages massive flow states.

Packet loss. Figure 14(a) illustrates the packet loss rate as the flow scale expands. Both the Tofino-based gateway and Gryphon maintain zero packet loss throughout the sweep, proving their ability to handle 10M concurrent flows stably. In contrast, the software-only gateway suffers a significant performance collapse once the flow scale exceeds 5M. Beyond this point, the overhead of managing high-density flow tables leads to packet drops exceeding 5% at peak scale, highlighting the limitations of software-based forwarding when flow states exceed the capacity of local CPU caches.

End-to-end latency. We evaluate latency within the stable operating region (up to 4M flows) where all candidates exhibit zero packet loss, which allows for a fair comparison of the steady-state forwarding performance. The results (Figures 14(b)–14(c)) show that Gryphon substantially reduces both average and P99 latency compared to the software-only gateway. At a flow scale of 4M, Gryphon reduces the average latency by 68.7% and the P99 latency by 79.3%. In addition, Gryphon adds only a modest 8μs delay relative to the Tofino-only design. The near-flat latency profile in Figure 15 confirms that Gryphon provides the deterministic performance needed for latency-sensitive services, effectively avoiding the jitter and timeouts typically seen in software solutions under heavy flow loads.

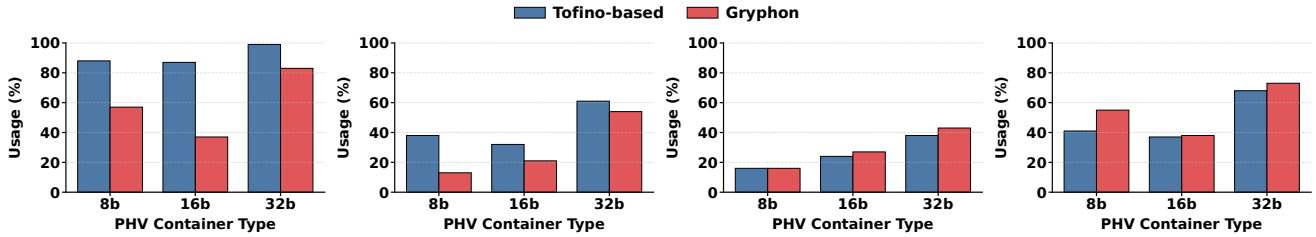
Per-Function DPU Overhead. We isolate the per-packet overhead introduced by each inline function on the DPU by measuring average, P99, and P99.9 latencies across five pipeline configurations and three flow scales (10K, 100K, 1M), as shown in Figure 16.

Flow Hit—in which matching flows are forwarded entirely within the DPU hardware pipeline—serves as the baseline for this breakdown.

Software upcall overhead. When a packet falls back to the software upcall path, the average latency rises to ~40 μs and the P99.9 latency exceeds 60 μs, representing ~14 μs (1.5×) and ~30 μs (2×) overhead over *Flow Hit*, respectively. This gap is likely influenced by overheads in hardware–software transitions.

Flow-logging overhead. Enabling per-flow logging (*Hit+Log*) appends a small number of counter writes to the existing hardware pipeline. Compared to *Flow Hit*, this adds less than 0.1 μs on average and less than 0.2 μs at P99.9 across all flow scales, indicating that flow logging is effectively free at the hardware forwarding level.

Inline-encryption overhead. Further enabling encryption and decryption (*Hit+Log+Crypto*) leverages the DPU's dedicated on-chip crypto engines. At 100K and 1M flows, the additional overhead beyond *Hit+Log* remains within 0.2–0.3 μs on average and under 0.3 μs at P99.9. At 10K flows, we observe a higher tail-latency spike, while the average overhead remains modest at +0.1–0.2 μs. This suggests that inline crypto is lightweight in the common case, with occasional tail events at smaller flow scales. Across hardware-offloaded configurations, average latency grows by less than 1 μs as the flow count scales from 10K to 1M, demonstrating stable performance under increasing flow pressure.



(a) Pipeline 0 (b) Pipeline 1 (c) Pipeline 2 (d) Pipeline 3

Figure 17: PHV usage across pipelines (the worst case).

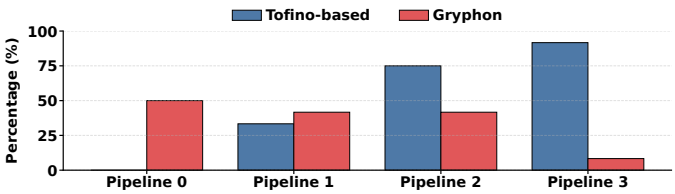


Figure 18: SRAM saturation percentage across pipelines.

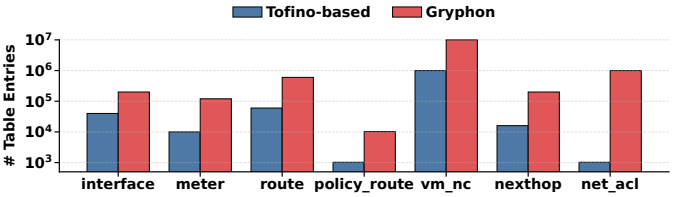


Figure 19: Table capacity of different versions.

Table 4: Comparison of efficiency metrics (normalized to best-in-class = 1.00×).

Cloud Gateway	Table capacity	Throughput (Gbps)	Per device		Efficiency (Normalized, 1.00× is Best)				Space
			Power	Cost	Power/ Tbps	Power/ Entry	Cost/ Tbps	Cost/ Entry	
Software-only	> 10×	~200	1.11×	1.00×	**8.89**×	1.00×	**6.67**×	1.00×	2U
Tofino-based	1×	~1600	**1.00**×	1.20×	1.00×	**>9.00**×	1.00×	**>12.00**×	2U
Gryphon	> **10**×	~ **1600**	2.08×	3.64×	2.08×	1.87×	3.03×	~3.64×	2U

8.3 Hardware Efficiency and Scalability

In this section, we evaluate the hardware efficiency of Gryphon by comparing its resource usage and table capacity against the Tofino-based gateway.

PHV utilization. Figure 17 compares PHV utilization for the Tofino-based cloud gateway and Gryphon. The results show that 32-bit PHV constrains the scalability of the Tofino-based gateway: in the worst case, pipeline 0 nearly exhausts its 32-bit PHV capacity. In contrast, Gryphon maintains a safe scaling margin across all three PHV widths (8/16/32-bit) and pipelines; in every pipeline, utilization for each width remains below 85%, enabling further scaling.

SRAM utilization. We define stages with SRAM usage > 95% as saturated, and plot the percentage of such stages for each pipeline in Figure 18. For the Tofino-based gateway, over 90% of stages (11 out of 12) in pipeline 3 are saturated, whereas for Gryphon the fraction of saturated stages stays below 60% in every pipeline. The relatively higher utilization observed in a few stages is caused by compiler placement and resource packing after adding metadata and steering logic, rather than by a new capacity bottleneck. Overall, Gryphon's SRAM utilization remains healthy, providing a substantial margin for growth and reducing the risk of resource scarcity.

Table capacity. Figure 19 summarizes the maximum table capacities for the Tofino-based cloud gateway and Gryphon. Leveraging DPUs' massive memory, Gryphon supports substantially more entries — up to 10M `gw_vm_nc` entries versus 1M on the Tofino-based gateway. Gryphon also offers at least 10× more entries for `gw_interface`, `gw_meter`, `gw_route`, and `gw_nexthop`, and 1000× for `gw_net_acl`. In addition, Gryphon supports extra tables such as `gw_nexthop_affinity`, which are infeasible on the Tofino-based gateway.

Head-to-head comparison: trade-off. Our analysis reveals a shifting bottleneck across generations: (1) In the first stage, software gateways offered massive table capacity (> 10×) but were severely limited by throughput (~200 Gbps). (2) In the second stage, Tofino-based gateways resolved the throughput deficit (~1600 Gbps) but hit a bottleneck in table capacity. Simplistic scale-out incurs prohibitive overheads in efficiency: Scaling software to match Tofino's throughput requires roughly 8× the rack space, 8.89× power consumption and 6.67x procurement cost. While scaling switching ASICs to match software capacity results in unacceptable Power/Entry (> 9.00×), Cost/Entry (> 12.00×), and Space/Entry. Notably, Gryphon resolves this dilemma through high-density integration. By consolidating four DPUs into the 2U chassis, Gryphon achieves ~1600 Gbps throughput and > 10× table scale simultaneously. This density prevents the prohibitive Space/Entry costs, allowing Gryphon to serve not only massive public clouds but also space-constrained edge clouds [6], where physical rack space is a scarce and rigid resource.

Notably, Gryphon is not designed to outperform a Tofino-only gateway in pure line-rate forwarding. Instead, it preserves the

folded-pipeline line rate of ~1.6 Tbps while providing > 10× table capacity, substantially better power and cost per entry than scaling Tofino-only gateways, and a broader set of deployable gateway functions. Compared with our Tofino-only gateway, Gryphon enables seven additional functions, including fragmented-packet offload and encryption. Thus, cloud gateways need DPUs primarily for flexible processing and large state capacity, rather than for marginal raw-throughput gains over an already line-rate ASIC datapath.

9 Related Work

Cloud vendors have explored different network virtualization architectures. Azure offloads host networking to SmartNICs [36, 50] to reduce CPU overhead and improve performance. Google's Andromeda uses Hoverboard programming model, which uses gateways for the long tail of low bandwidth flows [51]. Recent work further explores disaggregated accelerator pools to absorb spillover network-function load beyond what a single host-attached accelerator can provide [52].

For centralized cloud gateway, the industry has witnessed a clear evolution in data plane technologies. Early deployments, such as those by Alibaba Cloud, relied on software-based gateways running on x86 server clusters to handle north-south and east-west traffic [6, 21]. However, due to CPU bottlenecks, these systems transitioned to programmable switching ASICs (e.g., Tofino) to achieve high throughput and cost efficiency [21]. Sailfish [21] represents this state-of-the-art switching ASIC-based approach, employing hardware-software co-design to mitigate the chip's limited on-chip memory. Luoshen [6] further advances this paradigm by building a hyper-converged programmable gateway for multi-service edge clouds. However, as cloud scale continues to explode, the strict capacity bounds of on-chip resources eventually limit these solutions [5].

To address this, other architectures explore FPGA-based acceleration. For example, Albatross [5] leverages FPGAs to implement packet-level load balancing (PLB) with containerized flexibility. Although solutions like Albatross offer larger table capacities than switching ASICs, FPGA development involves significantly longer development cycles compared to the P4-based workflow used in Gryphon. Besides, its limited per-device throughput (800Gbps) potentially becomes a bottleneck for bandwidth-hungry workloads in the petabit era. Similarly, Tiara [53] proposes a hardware-accelerated architecture for stateful layer-4 load balancing that combines a high-throughput programmable switch for traffic steering with FPGA/HBM-based processing for scalable per-flow state management. In contrast, Gryphon targets hyperscale multi-tenant cloud gateways rather than a specific stateful L4 load-balancing service: it supports a broader set of cloud-gateway functions, including tenant routing, ACLs, metering, VM–NC mapping, encapsulation/decapsulation, and optional inline functions such as flow logging and encryption.

By introducing DPUs into the pipeline, Gryphon overcomes the resource constraints of ASIC-only gateways like Sailfish and Luoshen, while avoiding the development complexity and higher costs associated with FPGA-based designs like Albatross. This positions Gryphon as a viable and efficient candidate for next-generation cloud infrastructures.

10 Lessons Learned

Mitigating hash-induced DPU hotspots. Initially, we assumed that hash-based steering [2] would be sufficient to balance load across DPUs; however, deployment traffic skew [32, 33] repeatedly concentrated heavy-hitter flows on the same DPU. To address this, we implemented two complementary modes for steering traffic across DPUs. By default, packets are steered to DPUs by hashing flow metadata to select a DPU-facing Tofino port. Under hash skew, Gryphon switches to one-to-one mapping, where selected traffic groups are pinned to specific DPU-facing ports. This port redirection facilitates flow rebalancing, helping the inter-DPU load reach a balanced state. We found the two modes complementary in operations, with hashing in the common case and deterministic pinning activated only during skew episodes.

DPU-aware offloading. Although DPUs significantly extend the storage capacity and programmability of switching ASICs, our experience shows that naively migrating dataplane functions to the DPU can significantly degrade forwarding performance if architectural placement and implementation are not carefully designed. **Case 1 (Prefix matching on DPU):** Initially, we followed the common design intuition [16, 38] that the DPU's massive off-chip DRAM could be effectively utilized to host large prefix tables and implement LPM. However, LPM requires multiple dependent probes per packet, turning each lookup into iterative off-chip DDR accesses. Although lookups can be served from fast on-chip SRAM, the added DDR accesses quickly dominated processing latency, significantly degrading performance. Therefore, we moved prefix matching back to the switching ASICs and implemented it using Tofino's ALPM in on-chip SRAM [21]. **Case 2 (metering on DPU):** We implemented a token-bucket meter on the DPU for rate limiting. Our initial meter implementation used a compact fused arithmetic expression for token refresh, but this caused the compiler to generate expensive wide-operand arithmetic, resulting in more than 60 instructions. By rewriting the computation in a compiler-aware form and keeping intermediates narrow, we reduced the meter to about 40 instructions.

Scaling processing-heavy offloads. A lesson from Gryphon is that the DPU tier should be treated not only as expanded memory for large tables, but also as an elastic processing tier. While our current deployment is primarily driven by table-capacity pressure, many gateway features can become compute-bound as operators add richer inline functions. Compute-heavy operations such as encryption and compression/decompression are therefore better placed on DPU accelerators or programmable DPU pipelines than on switching ASICs. More importantly, Gryphon's multi-DPU steering makes such processing capacity scalable: traffic can be spread across stronger DPUs or across more DPU cards per node without changing the logical control-plane abstraction. This design makes Gryphon applicable not only to memory-bound gateways, but also to processing-bound network functions.

P4Bridge fits best in gateway designs with 5-tuple-based fast/slow path separation. In our deployment, we find that P4Bridge aligns well with pipelines that use 5-tuple matching and implement fast/slow path selection. In such cases, due to the constraints on the maximum match key size of a single P4 table, a

single control-plane logic table is typically decomposed into multiple dataplane tables, resulting in a natural 1:n mapping that reflects the pipeline's structural constraints. By contrast, in designs which use ASIC-only forwarding and non-5-tuple-based matching—logic tables tend to follow a simpler 1:1 mapping with dataplane tables. This simplicity may weaken P4Bridge's value in practice. Although more aggressive table merging (i.e., m:1 mappings) improves hardware resource utilization, the resulting increase in development complexity may undermine the design goal of P4Bridge of accelerating development. As a result, we adopt the simpler 1:1 mapping in non-5-tuple-based cases, which maintains the clarity of the logical API model and avoids overcomplicating the P4Bridge's development.

DPUs for faster iteration and debuggability than FPGA offload. We initially explored an FPGA-based offload [5] design for the same class of gateway functions. While FPGAs offer strong programmability and can host large tables, we found the end-to-end engineering loop substantially slower in practice: changes often span RTL, drivers, host software, and simulation/emulation [54, 55], and a single feature iteration can take on the order of half a year from design to validated deployment. Such long cycles make timely delivery difficult for us in a fast-moving production environment. In contrast, vendor DPUs provide a more integrated hardware/software stack and mature tooling, which improves iteration velocity and debuggability for production rollout [16, 22]. It motivated our decision to prioritize DPU-based offloading for productization while keeping the control-plane-facing abstractions stable to accommodate future hardware changes.

P4Bridge stabilizes control-plane interfaces during iterations. Due to strict hardware constraints and performance objectives, the implementation of the DPU pipeline is often deeply coupled with physical resources. Architectural evolution——whether for refactoring, optimization, or introducing new features such as encryption and flow logging——necessitates reshaping the physical table layout (e.g., splitting or merging P4 tables to combine or decompose functionality during refactoring or when adding modules such as flow logging and encryption). Exposing these pipeline- and vendor-specific table APIs to control-plane developers creates brittle dependencies: bottom-up layout changes repeatedly force control-plane code changes. Our operational experience confirms that Gryphon (P4Bridge) shields upper layers from this churn. Across multiple production iterations (e.g., P4 table adjustments and adding flow logging/encryption), the logical pipeline exposed to the control plane remained stable. Moreover, during our migration from BlueField [56] to Pensando, P4Bridge successfully masked hardware- and vendor-specific differences across DPUs. Adapting the abstraction layer required substantially less engineering and collaborative effort than rewriting upper-layer control logic against the DPU's native interface.

Future-proofing the dataplane across switching ASIC generations. We initially built Gryphon on Intel Tofino, taking advantage of its native P4 match-action abstraction [19]. At hyperscale, however, hardware churn is the norm: as deployments expand and the ASIC ecosystem evolves, we must be able to retarget the dataplane to other switches (e.g., Broadcom Trident 5 [57]) without a ground-up rewrite. To this end, we implement a P4-like dataplane in NPL [57]. Since NPL's table-side expressiveness differs from Tofino's per-entry action binding, we adopt a disciplined convention: each P4 action is encoded as an *action id*, its parameters are exported as metadata fields, and a hardware FSL (Flexible Switch Logic) block—supporting predicate evaluation and arithmetic—dispatches and executes the corresponding behavior. Based on our engineering experience with the Broadcom Trident 5, we have found that the P4-like NPL substantially reduces porting effort and shortens iteration cycles; combined with P4Bridge, it also keeps control-plane changes minimal when retargeting to different switching ASICs. Appendix A presents a concrete example.

Gryphon's general applicability. Initially, the community's consensus has often focused on scaling out ASIC-only clusters to meet growing demands [1, 6, 21]. However, in our environment, petabit-scale aggregate traffic and explosive growth of forwarding rules push both software gateways and ASIC-only designs to their limits. From the perspective of small and medium-sized cloud operators, simpler designs may be "good enough" today—many deployments may not yet reach hyperscale table pressure or require a scaling architecture. But as workloads grow and multi-tenant isolation demands more state, the gap between required table capacity and what on-chip resources can provide will widen; moreover, scaling out ASIC-only gateways fragments state and complicates traffic steering and synchronization. We therefore believe the design space Gryphon targets — tight ASIC–DPU coupling for "capacity + performance" rather than a pure software/ASIC approach—is not a niche solution, but a direction that more operators will likely need sooner or later [5, 38, 44].

11 Conclusion

Gryphon is ByteDance's third-generation cloud gateway. DPU integration, together with the hierarchical co-offloading strategy, overcomes challenges and limitations of previous gateways, without compromising their core strengths. Built for the petabit era, Gryphon scales to support massive traffic workloads while meeting emerging business needs. We are continuously optimizing this new architecture to deliver higher performance and meet emerging business demands. As we move forward with the system's continuous upgrades and deployments, we are committed to sharing our operational experiences and insights with the cloud networking community.

Acknowledgement

We would like to thank the anonymous reviewers and the shepherd for their constructive comments. This work was supported by the National Key Research and Development Program of China under Grant No. 2024YFB2906602, and in part by the National Natural Science Foundation of China (NSFC) (No. 62372009). Generative AI tools were used to assist with language polishing and grammatical improvements.

References

[1] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hong Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review*, 45(4):183–197, 2015.

[2] Daniel 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 *Nsdi*, volume 16, pages 523–535, 2016.

[3] Bytedance volcengine. https://www.volcengine.com/.

[4] Bytedance tiktok. https://www.tiktok.com/.

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

[6] Tian Pan, Kun Liu, Xionglie Wei, Yisong Qiao, Jun Hu, Zhiguo Li, Jun Liang, Tiesheng Cheng, Wenqiang Su, Jie Lu, Yuke Hong, Zhengzhong Wang, Zhi Xu, Chongjing Dai, Peiqiao Wang, Xuetao Jia, Jianyuan Lu, Enge Song, Jun Zeng, Biao Lyu, Ennan Zhai, Jiao Zhang, Tao Huang, Dennis Cai, and Shunmin Zhu. Luoshen: A hyper-converged programmable gateway for multi-tenant multi-service edge clouds. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*, pages 877–892. USENIX Association, 2024.

[7] Bytedance doubao. https://www.doubao.com/.

[8] Anirudh Sivaraman, Thomas Mason, Aurojit Panda, Ravi Netravali, and Sai Anirudh-Kondaveeti. Network architecture in the age of programmability. *Comput. Commun. Rev.*, 50(1):38–44, 2020.

[9] Nhu-Ngoc Dao, Anh-Tien Tran, Ngo Hoang Tu, Tran Thien Thanh, Vo Nguyen Quoc Bao, and Sungrae Cho. A contemporary survey on live video streaming from a computation-driven perspective. *ACM Comput. Surv.*, 54(10s):202:1–202:38, 2022.

[10] JIAXIN LEI. Enhancing the efficiency and scalability of cloud networking systems. 2024.

[11] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference, IMC 2017, London, United Kingdom, November 1-3, 2017*, pages 78–85. ACM, 2017.

[12] Francesco Fusco and Luca Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, pages 218–224. ACM, 2010.

[13] Benjamin H Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. 2010.

[14] Quan Chen, Shuai Xue, Shang Zhao, Shanpei Chen, Yihao Wu, Yu Xu, Zhuo Song, Tao Ma, Yong Yang, and Minyi Guo. Alita: comprehensive performance isolation through bias resource management for public clouds. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 32. IEEE/ACM, 2020.

[15] Zerui Guo, Jiaxin Lin, Yuebin Bai, Daehyeok Kim, Michael M. Swift, Aditya Akella, and Ming Liu. Lognic: A high-level performance model for smartnics. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2023, Toronto, ON, Canada, 28 October 2023 - 1 November 2023*, pages 916–929. ACM, 2023.

[16] Elie F. Kfoury, Samia Choueiri, Ali Mazloum, Ali AlSabeh, Jose Gomez, and Jorge Crichigno. A comprehensive survey on smartnics: Architectures, development models, applications, and research directions. *IEEE Access*, 12, 2024.

[17] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. TEA: enabling state-intensive network functions on programmable switches. In *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*, pages 90–106. ACM, 2020.

[18] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. Characterizing off-path smartnic for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*, pages 987–1004. USENIX Association, 2023.

[19] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: programming protocol-independent packet processors. *Comput. Commun. Rev.*, 44(3):87–95, 2014.

[20] Michael Galles and Francis Matus. Pensando distributed services architecture. *IEEE Micro*, 41(2):43–49, 2021.

[21] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, Enge Song, Jiao Zhang, Tao Huang, and Shunmin Zhu. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 194–206, 2021.

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

[23] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 35–46, 2010.

[24] Jiang Wu, Hongbo Wang, Kun Qian, and Enmiao Feng. Optimizing latency-sensitive ai applications through edge-cloud collaboration. *Journal of Advanced Computing Systems*, 3(3):19–33, 2023.

[25] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, 2013.

[26] Sai Dikshit Pasham. Graph-based algorithms for optimizing data flow in distributed cloud architectures. *International Journal of Acta Informatica*, 1(1):67–95, 2022.

[27] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. In *Proceedings of the 17th ACM Symposium on Operating System Principles, SOSP 1999, Kiawah Island Resort, near Charleston, South Carolina, USA, December 12-15, 1999*, pages 217–231. ACM, 1999.

[28] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Geoffrey Iannaccone, Allan Knies, Mário S. Monteiro, and Sylvia Ratnasamy. Routebricks: Exploiting parallelism to scale software routers. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 15–28, 2009.

[29] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: A gpu-accelerated software router. In *Proceedings of the ACM SIGCOMM Conference*, pages 195–206, 2010.

[30] DPDK Project. Dpdk: Data plane development kit – enabling high performance packet processing on dpus. https://www.dpdk.org/, 2022.

[31] Luigi Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX Annual Technical Conference (ATC)*, pages 101–112, 2012.

[32] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*, pages 281–296. USENIX Association, 2010.

[33] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 503–514. ACM, 2014.

[34] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 15–28. ACM, 2016.

[35] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando A. Mujica, and Mark Horowitz. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*, pages 99–110. ACM, 2013.

[36] Daniel Firestone, Andrew Putnam, Sambrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian M. Caulfield, Eric S. Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert G. Greenberg. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 51–66. USENIX Association, 2018.

[37] Zeke Wang, Hongjing Huang, Jie Zhang, Fei Wu, and Gustavo Alonso. Fpganic: An fpga-based versatile 100gb smartnic for gpus. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 967–986. USENIX Association, 2022.

[38] Nathan Tibbetts, Sifat Ibtisum, and Satish Puri. A survey on heterogeneous computing using smartnics and emerging data processing units (expanded preprint).

CoRR, abs/2504.03653, 2025.

[39] Arjun Kashyap, Yuke Li, Darren Ng, and Xiaoyi Lu. Understanding the idiosyncrasies of emerging bluefield dpus. In *Proceedings of the 39th ACM International Conference on Supercomputing, ICS 2025, Salt Lake City, UT, USA, June 8-11, 2025*, pages 807–821. ACM, 2025.

[40] Sanjay Churiwala and I Hyderabad. Designing with xilinx® fpgas. In *Circuits &Systems*. Springer, 2017.

[41] Douglas Doerfler, Farzad Fatollahi-Fard, Colin MacLean, Tan Nguyen, Samuel Williams, Nicholas J. Wright, and Marco Siracusa. Experiences porting the su3_bench microbenchmark to the intel arria 10 and xilinx alveo U280 fpgas. In *IWOCL'21: International Workshop on OpenCL, Munich Germany, April, 2021*, pages 1:1–1:9. ACM, 2021.

[42] Yunkun Liao, Jingya Wu, Wenyan Lu, Hang Lu, Xiaowei Li, and Guihai Yan. SNO: securing network function offloading on fpga-based smartnics in untrusted clouds. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2025, Munich, Germany, October 26-30, 2025*, pages 1–9. IEEE, 2025.

[43] Klajd Zyla, Marco Liess, Thomas Wild, and Andreas Herkersdorf. Hipernoc: A high-performance network-on-chip for flexible and scalable fpga-based smartnics. In *Design, Automation & Test in Europe Conference, DATE 2025, Lyon, France, March 31 - April 2, 2025*, pages 1–7. IEEE, 2025.

[44] Daehyeok Kim, Vyas Sekar, and Srinivasan Seshan. Exoplane: An operating system for on-rack switch resource augmentation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1257–1272, 2023.

[45] Xiaoqing Sun, Xionglie Wei, Xing Li, Ju Zhang, Bowen Yang, Yi Wang, Ye Yang, Yu Qi, Le Yu, Chenhao Jia, Zhanlong Zhang, Xinyu Chen, Jianyuan Lu, Shize Zhang, Enge Song, Yang Song, Tian Pan, Rong Wen, Biao Lyu, Yang Xu, and Shunmin Zhu. Zooroute: Enhancing cloud-scale network reliability via overlay proactive rerouting. In *Proceedings of the ACM SIGCOMM 2025 Conference, SIGCOMM 2025, São Francisco Convent, Coimbra, Portugal, September 8-11, 2025*, pages 1251–1253. ACM, 2025.

[46] Henry Wang. Algorithmic longest prefix matching in programmable switch, December 17 2019. US Patent 10,511,532.

[47] Michel Hanna, Socrates Demetriades, Sangyeun Cho, and Rami Melhem. Progressive hashing for packet processing using set associative memory. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 153–162, 2009.

[48] Oliver Michel, Roberto Bifulco, Gábor Rétvári, and Stefan Schmid. The programmable data plane: Abstractions, architectures, algorithms, and applications. *ACM Computing Surveys (CSUR)*, 54(4):1–36, 2021.

[49] Wedge100bf-65x quick start guide, 2024. Rev. R01. Available at https://stordis.com/wp-content/uploads/2024/01/DCS802_Wedge100BF_64X_Quick_Start_Guide_R01.pdf.

[50] Daniel Firestone. VFP: A virtual switch platform for host SDN in the public cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 315–328. USENIX Association, 2017.

[51] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 373–387. USENIX Association, 2018.

[52] Deepak Bansal, Gerald DeGrace, Rishabh Tewari, Michal Zygmunt, James Grantham, Silvano Gai, Mario Baldi, Krishna Doddapaneni, Arun Selvarajan, Arunkumar Arumugam, Balakrishnan Raman, Avijit Gupta, Sachin Jain, Deven Jagasia, Evan Langlais, Pranjal Srivastava, Rishiraj Hazarika, Neeraj Motwani, Soumya Tiwari, Stewart Grant, Ranveer Chandra, and Srikanth Kandula. Disaggregating stateful network functions. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1469–1487, 2023.

[53] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, Xiongfei Geng, Tao Feng, Feng Ning, Kai Chen, and Chuanxiong Guo. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1345–1358. USENIX Association, 2022.

[54] Jeffrey Goeders and Steven J. E. Wilton. Signal-tracing techniques for in-system FPGA debugging of high-level synthesis circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 36(1):83–96, 2017.

[55] Sameh Attia and Vaughn Betz. Statemover: Combining simulation and hardware execution for efficient FPGA debugging. In *FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*, pages 175–185. ACM, 2020.

[56] Suhyeon Lee, Myoungsung You, Sooyeon Kim, and Taejune Park. Comprehensive performance analysis of security applications on the bluefield-3 smartnic. *Comput. Networks*, 275:111830, 2026.

[57] Broadcom Inc. StrataXGS Trident 5 (BCM78800). https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78800, n.d.

APPENDIX

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

A A Concrete Example of our P4-like NPL

In P4, a match-action table can directly bind multiple actions with distinct parameters. In NPL, we encode the action choice as an *action id* (e.g., `action_ctrl_0`) and export all action parameters as table fields, then implement an action-dispatch function in the hardware logic unit (FSL) to realize the corresponding behavior.

A.1 P4 (Tofino) Program Snippet

```
#define CONST_IP 0x0A0A0A0A

// action_hit
action fib_hit(inout switch_local_metadata_t local_md,
               switch_nexthop_t nexthop_index) {
  local_md.nexthop = nexthop_index;
  local_md.routed = true;
}

// action_1
action fib_myip_1(inout switch_local_metadata_t local_md,
                  switch_myip_type_t myip) {
  local_md.myip = myip;
  local_md.myflag = true;
}

// action_2
action fib_myip_2(inout switch_local_metadata_t local_md,
                  switch_myip_type_t myip) {
  local_md.myip = myip;
  local_md.myflag = false;
}

// action_3
action fib_myip_3(inout switch_local_metadata_t local_md,
                  bool myflag) {
  local_md.myip = CONST_IP;
  local_md.myflag = myflag;
}

// action_drop
action fib_drop(inout switch_local_metadata_t local_md) {
  local_md.routed = false;
  local_md.fib_drop = true;
}

table my_ipv4_host {
  key = {
    local_md.vrf : exact @name("vrf");
    local_md.lkp.ip_dst_addr[95:64] : exact @name("ip_dst_addr");
  }
  actions = {
    fib_hit;
    fib_myip_1;
    fib_myip_2;
    fib_myip_3;
    fib_drop;
  }
  const default_action = fib_drop(local_md);
  size = my_host_table_size;
}

control SwitchIngress(...) {
  my_ipv4_host.apply();
  ...
}
```

A.2 P4-like NPL Implementation

```
P4 logical_table my_ipv4_host {
  table_type : hash;
  minsize    : my_host_table_size;
```

```
  maxsize    : my_host_table_size;

  keys {
    bit[FIELD_32_WD] ipv4_addr;
    bit[VRF_WD]      vrf;
  }

  // Export action choice + all potential parameters as fields.
  fields {
    bit[FIELD_4_WD]  action_ctrl_0;
    bit[FIELD_16_WD] nexthop_index;
    bit[FIELD_32_WD] myip;
    bit[FIELD_1_WD]  routed;
    bit[FIELD_1_WD]  myflag;
  }

  key_construct() {
    if (_LOOKUP0 == 1) {
      vrf       = ing_obj_bus.vrf;
      ipv4_addr = pkt_fwd_field_bus.ip_hdr_dip[31:0];
    }
  }

  fields_assign() {
    if (_LOOKUP0 == 1) {
      if (_VALID == 1) {
        ing_obj_bus.nexthop_index = nexthop_index;
        ing_obj_bus.myip_0 = myip[31:16];
        ing_obj_bus.myip_1 = myip[15:0];
        ing_cmd_bus.routed = routed;
        ing_cmd_bus.myflag = myflag;
        ing_cmd_bus.action_ctrl_0 = action_ctrl_0;
      } else { // miss
        ing_obj_bus.nexthop_index = 0;
        ing_obj_bus.myip_0 = 0;
        ing_obj_bus.myip_1 = 0;
        ing_cmd_bus.routed = 0;
        ing_cmd_bus.my_fib_drop = 1;
        ing_cmd_bus.myflag = 0;
        ing_cmd_bus.action_ctrl_0 = 0;
      }
    }
  }
}

// Action dispatch implemented in the FSL logic unit.
function ifsl_my_ip_host_process() {
  if (ing_cmd_bus.action_ctrl_0[1:1] == 1) {
    ing_cmd_bus.myflag = 1;
  } else if (ing_cmd_bus.action_ctrl_0[2:2] == 1) {
    ing_cmd_bus.myflag = 0;
  } else if (ing_cmd_bus.action_ctrl_0[3:3] == 1) {
    ing_obj_bus.myip_0 = 0x0A0A;
    ing_obj_bus.myip_1 = 0x0A0A;
  }
}

function INGRESS_PROCESS() {
  my_ipv4_host.lookup(0);
  ifsl_my_ip_host_process();
}
```

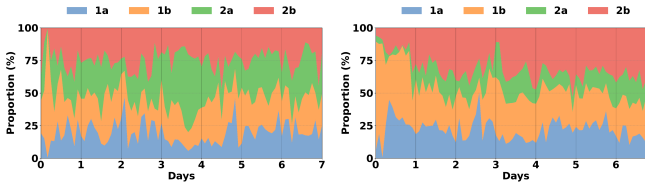
B Supplementary Experimental Results

B.1 DPU Traffic Distribution

We provide supplementary traffic distribution across different deployment instances in Figure 20 and 21. These results demonstrate consistent characteristics: intra-DPU ports remain susceptible to imbalance and the traffic across DPUs is evenly distributed. This further validates the robustness and effectiveness of our dual-mode steering strategy across various production nodes.

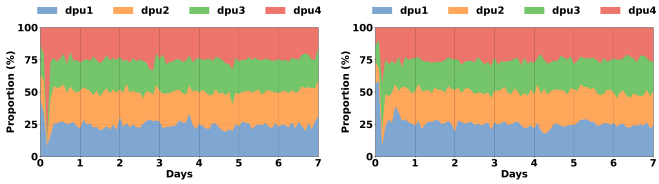
B.2 Detailed SRAM Usage

We present the detailed per-stage SRAM usage for each pipeline in Figure 22. The results show that the Tofino-based gateway exhausts nearly all SRAM capacity across almost every stage in Pipeline 3, creating a rigid resource bottleneck that severely limits further scalability.



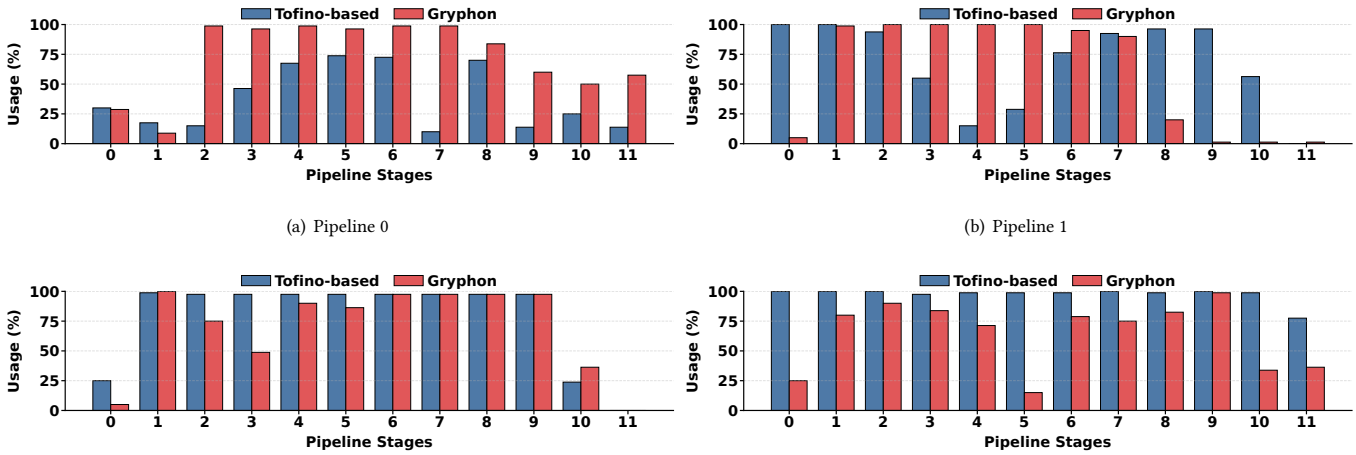
(a) DPU 1 (b) DPU 2

Figure 20: Intra-DPU traffic distribution.



(a) Node 1 (b) Node 2

Figure 21: Inter-DPU traffic distribution.



(a) Pipeline 0 (b) Pipeline 1

(c) Pipeline 2 (d) Pipeline 3

Figure 22: SRAM usage over stages across pipelines.