CStar Gateway: Augmenting Public Cloud Infrastructure for Heterogeneous Network Function Virtualization

Paper #14, 12 pages body, submitted to Operational Systems Track

Abstract

Major cloud providers often build NFV products by reusing existing tenant-oriented architectures. These designs were originally developed and optimized for tenant virtual machines (VMs) that operate at network endpoints. In contrast, network function (NF) VMs function as intermediate forwarding nodes shared by multiple tenants, which have very different resource demands. Directly applying tenant-oriented designs creates mismatches between infrastructure capabilities and NF requirements. A common case arises when NF VMs serve many tenants: they typically exhaust vNIC and session entries on vSwitches well before CPU resources are saturated. At this point, NF VMs cannot accept additional traffic despite having idle CPU cycles. Providers usually address this by scaling out or scaling up NF VMs, which wastes resources and increases cost. This is just one example, where numerous other mismatches remain. To address these, we present CStar Gateway. CStar Gateway shifts NF VM multi-tenancy support from vSwitches into NF VMs, which reduces vNIC and session bottlenecks with minimal changes to the existing cloud infrastructure. CStar Gateway also identifies and redirects I/O- or CPU-intensive flows to FPGA-based NFs, increasing the service capacity. In addition, CStar Gateway takes over NF VM elasticity support from vSwitch, simplifying and accelerating the scaling process, and thereby enhancing overall system flexibility. Deployment results show that the design improves CPU and I/O utilization of NF VMs by at least 5x and reduces NF cluster capital expenditure by 80.43% to 91.34%.

1 Introduction

Network Function Virtualization (NFV) enhances service flexibility, scalability, and efficiency in public clouds by decoupling Network Functions (NFs) from proprietary hardware, thereby reducing vendor lock-in and improving resilience [24]. In contrast to stateless forwarding, NFV instances often need to manage state while sustaining line-rate performance, which presents challenges for both scale-up at the instance level [9, 18, 19, 28] and scale-out across multiple instances [12, 20, 21, 26]. Furthermore, in public clouds, tenant traffic exhibits high variability, and service level agreements (SLAs) differ across tenants. Consequently, NFV systems must incorporate sufficient elasticity to accommodate traffic surges while ensuring strict inter-tenant isolation [14, 15, 25].

To address challenges, cloud service providers have reported numerous NFV implementations based on their design goals and infrastructure legacies [3, 5, 7, 16, 27]. They maximize reuse of existing infrastructure to avoid redundant development. For example, Azure integrates basic NF capabilities like ACLs in its virtual switch (vSwitch) [7], and builds a DPU-accelerated shared resource pool to handle overflow traffic exceeding the vSwitch's capacity [3]. In contrast, AWS Hyperplane [16] and Alibaba CyberStar (*a.k.a.*, CStar) [27] use standard VMs to implement NFs, enabling more complex capabilities (*e.g.*, L7-Load Balancer (LB), VPN). These NF VMs fully leverage existing cloud infrastructure for elasticity and fault tolerance. Moreover, NF VMs are decoupled from the vSwitch, ensuring that sudden traffic surges or failures in NF VMs do not compromise core VPC routing.

CStar is a multi-tenant, multi-service cloud-based NFV architecture. In CStar, VMs of the same specifications provided to tenants [2] are leveraged to accommodate NFs. These NF VMs can coexist with tenant VMs on the same vSwitch and can also migrate freely within the cloud for high availability and resource efficiency. CStar supports scaling out multiple NF VMs to serve a single tenant demand, or sharing a single NF VM among service demands from different tenants to improve resource efficiency. To preserve isolation across tenants' demands in a shared environment, CStar provisions multiple vNICs on a single NF VM, each carrying NF's demand-specific traffic throttling configurations, and routes traffic from different tenants' demands through separate vNICs to the NF processing logic.

Despite the advantages of CStar, we still encounter several technical issues. The first is that some NF VMs exhaust the vNICs or sessions provided by vSwitch, even though their CPU or I/O utilization remains low. This prevents them from

accepting additional traffic. As a result, we are forced to provision more NF VMs to meet growing traffic demands, which unreasonably increases our costs. In such cases, resource utilization remains low for all but the bottlenecked resources. Our analysis shows that the root cause lies in the mismatch between the heterogeneous resource requirements of NF VMs (*e.g.*, vNICs, sessions) and the resource capabilities offered by vSwitches to VMs. Since tenant VMs constitute the majority of cloud workloads, cloud infrastructure is optimized for them. However, NF VMs differ significantly from tenant VMs in two aspects: (1) tenant VMs typically require only 1-2 vNICs, while NF VMs often need hundreds of vNICs as network intermediate nodes; (2) tenant VMs consume session resources only for their own traffic, whereas NF VMs, positioned at traffic aggregation points, must maintain sessions for a large number of concurrent flows.

The second issue lies in the imbalance between I/O and CPU resource utilization within individual NF VMs. Some NF VMs exhibit high I/O utilization but low CPU usage, while others frequently suffer from CPU overload due to heavy-hitter traffic. This disparity arises from the different packet processing models of different NFs: some are I/O intensive (*e.g.*, Transit Router (TR) [1]), whereas others are more CPU-intensive (*e.g.*, VPN). As a result, NF VMs consume resources in varying proportions when processing traffic.

The third issue is that the elasticity support of NF VMs differs significantly from that of tenant VMs. Each tenant VM has a unique IP, so scaling out tenant VMs only requires assigning new IPs via the controller for direct access to the newly created VMs. In contrast, NF VM scaling is hidden behind a service IP assigned to the tenant, with traffic distributed to NF VMs via vSwitch hashing. Since NF VMs often maintain state, consistent hashing [6,13] on the vSwitch is preferable to reduce state inconsistency during scaling. This adds significant performance overhead to the vSwitch.

To address these issues without causing changes to the existing cloud infrastructure, we propose a "minimally invasive" solution by introducing a programmable gateway named CStar Gateway that sits between tenant VMs and NF VMs. To eliminate vNIC exhaustion, the gateway encapsulates an NF-specific configuration identifier (*i.e.*, fake vNIC ID) into an extra GENEVE tunnel [11] and delivers the identifier to the remote NF VM. The NF VM extracts the identifier and executes the corresponding traffic throttling operations (*e.g.*, pass, meter, or drop) that were previously handled by the vSwitch. As a result, NF VMs no longer need multiple vNICs from the vSwitch to differentiate NF-specific demands for different tenants. To address session exhaustion, the gateway relays traffic from tenant VMs and aggregates multiple flows with distinct 5-tuples into a single flow by encapsulating them into the GENEVE tunnel. This enables the vSwitch to track only one session per NF VM. To address the I/O-CPU imbalance in NF VMs under bandwidth-heavy and elephant flows, we offload these workloads to FPGAs, whose resource profile better matches their demands. To reduce the burden on the vSwitch in supporting the elasticity of NF VMs, we also offload the consistent hashing functionality from the vSwitch to the gateway. So far, CStar Gateway has been used for about two years. We share the experiences and lessons learned from its development and deployment.

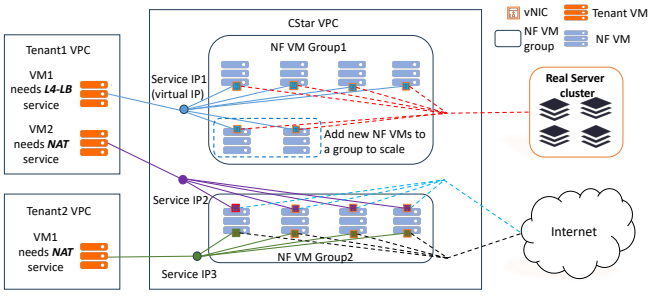

Figure 1: Cloud-based NFV deployment with CStar.

Our major contributions are summarized as follows.

- We present CStar Gateway, a minimally invasive architecture that shifts multi-tenant, multi-demand support from the vSwitch to NF VMs using tunnel encapsulation, without modifying the vSwitch. This removes vNIC and session limits, enabling NF VMs to fully utilize idle CPU and I/O resources, improving per-NF VM resource utilization by up to 5×.
- We design a hybrid data plane that offloads CPU- and I/O-intensive flows to FPGA for line-rate performance, while keeping other flows on elastic NF VMs. This approach combines the FPGA's high throughput with VM's elasticity.
- We offload NF VM elasticity support from vSwitch to CStar Gateway, simplifying and accelerating the scaling process, enhancing overall system flexibility.
- After deployment across three availability zones (AZs), NF infrastructure costs are reduced by 80-91% by efficiently serving increased traffic with fewer NF VMs. With CStar Gateway, Alibaba Cloud is no longer plagued by the long-standing challenge of NFV profitability.

2 Cloud-based NFV Deployment

Multi-tenancy and Multi-demand NFs. As illustrated in Fig. 1, Alibaba Cloud's NF platform named Cstar serves multiple tenants with heterogeneous NF demands, comprising two dimensions: (1) NF type (*e.g.*, LB, NAT); (2) NF access control (traffic admitted only from designated source IPs, subject to rate limits). For example, as shown in Fig.1, Tenant 1 requires LB accessed only with VM1 and NAT accessed only with VM2; Tenant 2 has one NAT demand. As a network middlebox, the NF VM concurrently processes multi-tenant, multi-demand traffic. To ensure isolation, we provide stateful NF services at (tenant id, service id) granularity. Tenant id identifies the tenant, service id distinguishes between different NF demands. We combine vSwitch infrastructure support (*e.g.*, ACLs, metering) and in-VM programmable logic (*e.g.*,

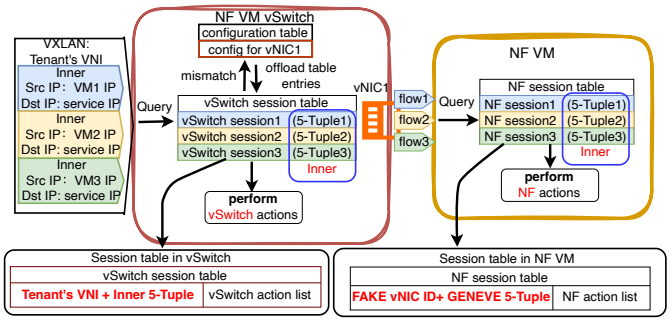

Figure 2: Role of session in vSwitch.

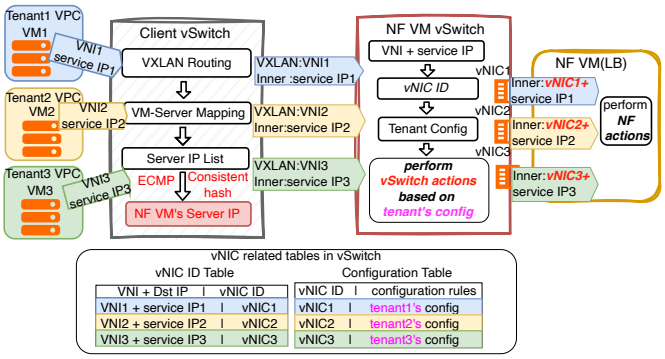

Figure 3: Packet journey from the tenant VM to the NF VM.

LB policy) to fulfill demands. For Tenant 1's LB demand: 1) vSwitch allows only VM1's IPs; 2) NF VM distributes traffic via least-connections. Due to the large scale, a single NF VM is insufficient. We deploy an NF VM group to share the traffic load of a tenant demand. Tenants access services via a virtual Service IP. Traffic is distributed across the group via ECMP. This NF VM group architecture provides rapid elasticity: NF VMs can be dynamically added or removed to match traffic without service interruption.

Tenant NF Access Control. In Alibaba Cloud's architecture, each VM (including NF VM) is assigned one or more vNICs. Each vNIC holds specific configurations, such as security group rules (*e.g.*, allow 10.1.0.0/16) and rate limits (*e.g.*, 1 Gbps). vSwitch is applied to process incoming traffic (pass/drop/meter) before forwarding to the VM. To serve Tenant 1's LB demand: 1) A dedicated vNIC is created on the NF VM's vSwitch; 2) Configured with Tenant 1's ACL and meter demands; 3) Routing steers all Tenant 1 LB traffic through this vNIC. Importantly, vSwitch handles only access control (ACL, metering). NF logic (*e.g.*, LB, NAT) is implemented inside the NF VM via programmable service code.

High-performance NF Traffic Processing. To ensure line-rate processing, every flow is assigned one session entry, created on its first packet, and reused for all subsequent packets. As shown in Fig. 2, the vSwitch looks up the target vNIC and its configs (ACL, metering) and creates a session entry storing: 1) vNIC ID (where to steer); 2) precomputed actions (pass/drop/meter) on the first packet. Later packets match only the session and perform cached actions to skip configuration lookup and compute actions. This design embodies the fast/slow-path separation principle. NF VMs also use session design. But the configs and session entries in NF VM are different from those in vSwitches. In CStar architecture, no NF packet is forwarded without a session, ensuring all traffic runs at line rate.

Routing of Tenant NF Traffic to NF VMs. To illustrate how Alibaba Cloud uses multi-vNIC in vSwitch to deliver multi-tenant and multi-demand NFs, we trace the end-to-end path from tenant traffic generation to NF VM processing (Fig. 3). When three tenants (Tenant 1–3) each have a distinct NF, the cloud provider allocates a dedicated Service IP to each. NF VMs share the same underlying infrastructure as tenant VMs, and their traffic is routed in the same way. In Client-side vSwitch, when receiving a packet, it first checks the VXLAN routing table. If the destination matches a Service IP, the packet is mapped to an internal cstar VPC (invisible to tenants), and the tenant's VNI is preserved. The vSwitch then looks up the target NF VM group using (VNI, Service IP) and returns the list of backend server IPs hosting NF VMs in that group. Finally, it applies ECMP on the packet's inner 5-tuple to select a backend server and forwards the packet to the corresponding NF VM host.

When receiving a packet, the NF-side vSwitch extracts (Tenant VNI, Service IP) from the packet, looks up the vNIC ID in the vNIC ID table, and retrieves the corresponding vNIC (*e.g.*, vNIC1 for Tenant 1's demand). It then fetches the tenant-specific configuration (security group, rate limit) from the configuration table based on the vNIC ID (*e.g.*, tenant1's config), and conducts per-tenant NF access control (pass/drop/meter/decap) accordingly. The packet is then delivered through the selected vNIC to the NF VM, which executes the NF service logic (*e.g.*, SNAT/DNAT, LB).

Elastic NF Management. In CStar, a controller creates and manages NF VM resources. Upon receiving a tenant demand for creating an NF service, it assigns a group of NF VMs to the tenant. To ensure high availability, the controller implements shuffle sharding [17] to distribute NF VMs across different servers, reducing the risk of "query of death" [22]. After resource allocation, the controller configures routing tables in the vSwitches, managing tenant VM traffic and NF VM traffic to ensure correct traffic delivery. When a tenant NF traffic increases, the controller can horizontally scale NF performance by adding more NF VMs to the group (Fig.1). Specifically, in CStar, "everything is a VM and a VM can go everywhere". A tenant vNIC can coexist with another tenant vNIC on the same NF VM, and NF VMs can coexist with tenant VMs on the same server. The elasticity of the cloud enables CStar to deliver high availability and resource efficiency.

3 NFV-Infrastructure Mismatches

Our cloud-based NFV reuses the existing public cloud infrastructure, including VMs, vSwitches, controllers, and physical servers. By doing so, it inherits the key benefits of cloud computing and avoids the cost of reinventing the wheel. However, the existing cloud infrastructure is primarily optimized for ten-

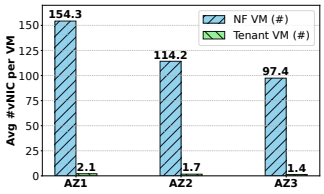

Figure 4: Average vNICs on NF VMs and tenant VMs.

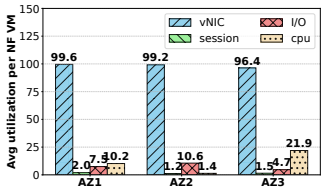

Figure 5: Resource utilization in VPC NAT NF VM.

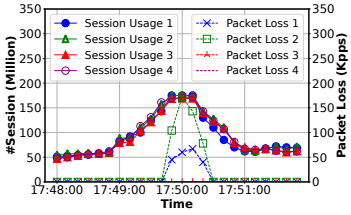

Figure 6: Packet loss due to vSwitch session exhaustion.

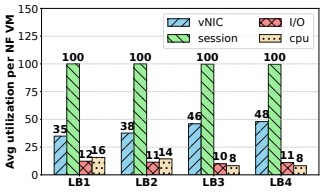

Figure 7: Resource utilization in LB NF VM.

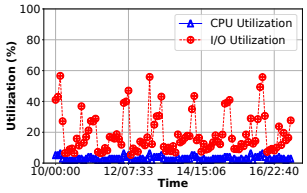

Figure 8: I/O saturation despite low CPU utilization.

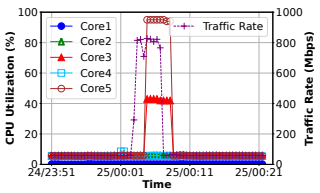

Figure 9: CPU core overload due to heavy-hitter flows.

ant VMs, which constitute the majority of workloads. Given that NF VMs make up a smaller fraction («1%, still thousands of VMs), the architecture has not been specifically optimized for their needs. However, since tenant VMs primarily handle compute tasks at network endpoints, while NF VMs serve as intermediate devices for traffic forwarding, their resource consumption patterns and architectural demands differ fundamentally. The current cloud infrastructure shows several mismatches with NFV requirements in both resource usage and performance.

To accommodate the broad range of tenant demands, Alibaba Cloud has introduced multiple specialized NF VM types (*e.g.*, Layer-4 LB, NAT, TR). These NF VM variants exhibit distinct resource consumption, directly correlated with the characteristics of the traffic they process. In the following sections, we analyze several representative NF VM types combined with their real-world service workloads. These case studies empirically reveal and quantify the resource mismatch between NF VMs and general-purpose tenant VMs.

Mismatch 1: NF VM's massive occupation of vNICs provided by vSwitch. In our production environment, we observe that NF VMs consume a substantially larger number of vNICs compared to general-purpose tenant VMs. As illustrated in Fig. 4, we measured the average number of vNICs per VM across three AZs, comparing NF VMs against typical tenant VMs. The results show that each NF VM, on average, consistently utilizes > 97 vNICs while tenant VMs typically use only ~2 vNICs.

Workload profiling shows that some NF VMs exhaust the vNICs allocated by the vSwitch. These NF VMs are primarily used to implement the VPC NAT. The VPC NAT resolves IP address conflicts when multiple tenant VPCs communicate with a shared service VPC. A common example is Alibaba Cloud's Simple Log Service (SLS). SLS runs in a dedicated, centrally managed VPC whose internal IP ranges frequently overlap with the private CIDR blocks used by tenant VPCs. VPC NAT NF VMs must bind one vNIC per tenant demand to maintain isolation. This quickly pushes them to the vNIC quota limit. In contrast, their workload (periodic, small log uploads with only NAT action) imposes negligible demand on NF VM's CPU, memory, I/O, and session tables allocated by vSwitch. As a result, vNICs become the bottleneck while other resources remain largely idle. We profiled VPC NAT NF VMs across three AZs. As shown in Fig. 5, all sampled NF VMs reached the vNIC allocation quotas of vSwitch, while other resources were grossly underutilized. Once vNICs are exhausted, NF VMs can no longer serve additional tenant demands even if other resources remain idle. Operators typically deploy more NF VMs as a workaround, which wastes resources and increases operational cost.

The vNIC quota per NF VM is constrained (*e.g.*, 400). This limit is not due to hardware incapacity, but to economic design. The memory on SmartNIC needs to be shared across network, storage, and other functions. As a result, the vSwitch deployed on a SmartNIC can support only about 4K vNICs per server. Because NF VMs share this vNIC pool with tenant VMs, allocating more to NF VMs would directly reduce tenant VM density and, in turn, per-server revenue. Quotas are therefore needed to ensure system-wide fairness and business efficiency. We experimented with network-optimized VMs, which provide more vNICs and higher bandwidth but with fewer CPU and memory resources to minimize waste. While these efforts help, NF VMs still hit the quota, albeit slightly later. We also chose not to design a custom VM for VPC NAT, given 1) high development cost, 2) limited market demand (normal tenants won't buy it), and 3) poor cost-effectiveness (NF VMs account for «1% of all VMs).

The vSwitch enforces fixed per-VM quotas on vNICs, sessions, and I/O at provisioning time. This static-allocation model ensures strong tenant isolation, predictable performance, and transparent billing. While it may occasionally lead to underutilization when workloads fall short of their reserved quotas, the expectation is that tenants select suitable VM flavors (*e.g.*, network-optimized instances) that align with their demands. The platform deliberately avoids dynamic or shared pools. Although such designs could improve average utilization, they introduce risks of resource hijacking, cross-tenant interference, and unpredictable performance, which outweigh their potential benefits in a commercial multi-tenant environment.

Mismatch 2: NF VMs require far more sessions than tenant VMs. In production, we frequently observe session table exhaustion alerts on NF VMs, directly triggering packet loss and service degradation. For example, in the 4-member NF VM group in Fig. 6, two instances reach 100% of their session capacity and experience severe drops, while the other two are nearing capacity and at imminent risk of failure. Session exhaustion occurs most commonly in NF VMs running LBs, which serve (1) microservices accessing backend storage and (2) large fleets of IoT devices connecting to VPC-hosted services (*e.g.*, storage, compute). Both workloads generate bursts of short-lived flows (often exceeding 1M concurrent sessions within seconds) that rapidly fill vSwitch session tables. Session exhaustion arises even when other resources remain underutilized: 1) I/O bandwidth stays low due to small request packets. 2) CPU remains largely idle as no L7 parsing is required, and 3) vNIC usage is moderate because we deliberately limit tenants per LB to preserve per-tenant quality. At this point, the session quota allocated by the vSwitch becomes the constraint on LB throughput. As shown in Fig. 7, all four LB NF VMs approach the session-capacity limit, while I/O, CPU, and vNIC remain substantially underutilized. These results confirm that session capacity, not bandwidth or compute, can become a dominant bottleneck.

Once the session table on the vSwitch is exhausted, new flows cannot be established, causing packet loss despite abundant CPU, memory, and I/O resources. The practical workaround is to scale out the NF VM group, which consumes additional resources and increases cost. Raising per-VM session quotas or introducing a special "high-session" VM type is infeasible for the same economic and architectural reasons that restrict vNIC scaling.

Mismatch 3: NF's I/O bandwidth can saturate despite low CPU utilization. Profiling shows some NF VMs operating near peak I/O utilization, while CPU usage barely reaches 6%. This imbalance demonstrates that the I/O can become the bottleneck on NF VMs. TR NF VM is a typical example, which forwards all traffic between interconnected tenant VPCs, carrying bandwidth-heavy workloads like HD video, real-time conferencing, and cloud storage sync. TR is stateless, L3-only forwarding logic, which involves no deep packet inspection, encryption, or L7 processing, without the need for high CPU utilization. Meanwhile, for cloud VMs, even I/O-optimized variants, their architecture is fundamentally advantageous for compute-intensive applications, leaving excess CPU capacity unused. This forces TR's NF VM scaling for I/O, leaving CPU idle, a direct consequence of a mismatch between a bandwidth-centric workload and a compute-centric VM model.

While both TR and VPC NAT interconnect multi-tenant VPCs, their roles differ fundamentally. TR forwards all traffic, generating high aggregate bandwidth, whereas VPC NAT only processes flows with IP conflicts. To maintain service quality, TR NF VMs also limit tenant density per instance. Unlike LB (which suffers from session explosion), TR traffic typically consists of long-lived, high-throughput flows, resulting in minimal session consumption. Consequently, TR faces moderate vNIC pressure (far less than VPC NAT) and negligible session pressure (unlike LB), making I/O bandwidth the major scaling constraint. Since NF VM pricing bundles CPU, I/O, and memory, operators pay for idle CPU cores when scaling for I/O, inflating operational cost. At Tbps-scale deployments, this inefficiency directly compresses profit margins, making current VM-based scaling economically unsustainable.

Mismatch 4: NF CPU can be overloaded by heavy-hitter flows. Within an NF VM, incoming packets (delivered via vNIC) are distributed across CPU cores using RSS. However, persistent elephant flows can be pinned to a single core, saturating it and causing packet loss for all flows processed on that core. These high-rate flows are unpredictable in timing and origin, and appear across NF types (including LB, VPC NAT, TR, and other stateful services), making it a universal risk. While vNICs are configured with metering policies to enforce tenants' demand-level bandwidth caps, these policies operate at the aggregate vNIC level and do not directly apply to each flow. Consequently, a single high-rate elephant flow can fully saturate a CPU core even though the vNIC's overall throughput remains within its configured limit.

Mismatch 5: NF VM elasticity is more complex than for tenant VMs. As noted Sec. 2, the vSwitch distributes tenant NF traffic across an NF VM group using ECMP hashing. When the group scales (VMs added/removed), the hash value changes, redirecting existing flows to different NF VMs. Since session state resides only on the original NF VM, the state must be migrated for flows to continue processing correctly. To mitigate disruption, consistent hashing is used on vSwitch to minimize flow remapping during scaling events. However, consistent hashing consumes significantly more vSwitch CPU and memory than standard hashing. In contrast, tenant VMs generally maintain stable 1:1 IP-to-VM mappings, requiring no hashing at all. Scaling an NF VM group triggers a control-plane storm. The controller must push updated forwarding rules to many distributed tenant-side vSwitches. This process is also very slow, taking 10s. As a result, NF scaling remains reactive, not predictive, and fails to meet the sub-second agility expected in cloud-native environments

4 NFV Overhead Outsourcing

4.1 Remove vNIC/Session Limits by Gateway

Aggregating Multi-tenant and Multi-demand vNICs. To overcome the vNIC limit, CStar Gateway aggregates all tenant demand-specific vNICs into a single vNIC per NF VM, regardless of the number of demands it serves. Without the gateway, the vSwitch stores tenant demand-specific configurations (*e.g.*, security groups, rate limits) on a per-vNIC basis and applies them during packet processing to enforce NF access control at the vNIC level. After vNIC aggregation, these configurations are offloaded to the NF VM itself. The

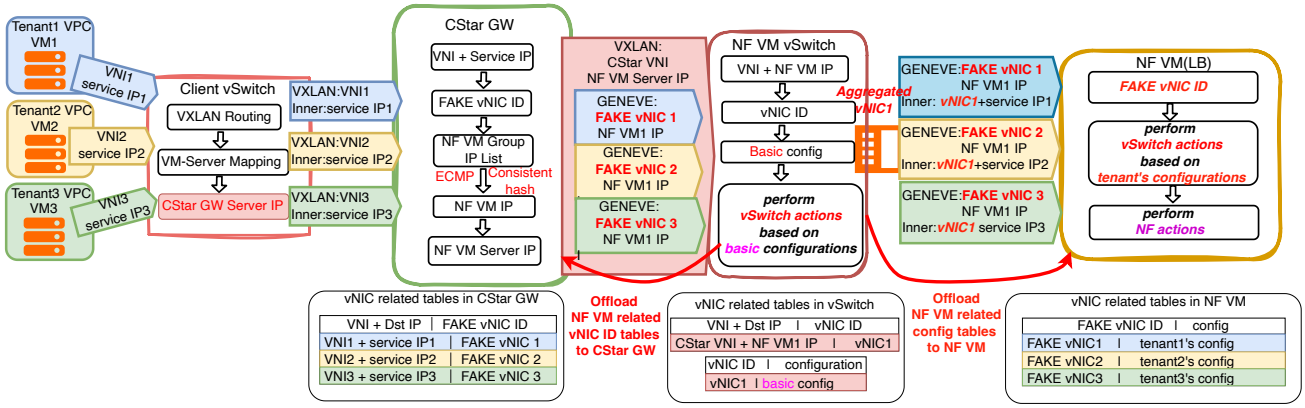

Figure 10: Packet journey from the tenant VM to the NF VM with CStar Gateway (GW) for vNIC aggregation at the NF VM.

vSwitch now uses a single lightweight vNIC, which carries only minimal and tenant-agnostic configurations (*e.g.*, basic ACLs and default meter tables).

NF VM must distinguish between configurations corresponding to multiple tenants with multiple demands (previously by using a vNIC ID corresponding to a physical vNIC in vSwitch). We introduce a logical identifier named fake vNIC ID. It is called "fake" because it serves only as a key to differentiate and index tenant demand-specific configurations inside the NF VM, with no corresponding physical vNIC. To preserve compatibility with the original logic, we retain the lookup mechanism based on the (VNI, Service IP) tuple, which previously resolved to a physical vNIC ID. The lookup table is offloaded to the CStar Gateway, which maps the tuple directly to the appropriate fake vNIC ID. To deliver this ID to the NF VM, the gateway encapsulates the original inner packet with a GENEVE header, embedding the fake vNIC ID in its option field, and then wraps it with an outer VXLAN header for transport across the underlay network.

With tenant demand-specific configurations (ACL, meter rules) migrated, the vSwitch can no longer conduct per-tenant demand NF access control. The NF access control function is now fully offloaded to NF VMs. As shown in Fig. 10, upon receiving a packet, the NF VM first parses the GENEVE header to extract the fake vNIC ID, retrieves the corresponding configurations, and executes vSwitch-equivalent actions before proceeding to NF logic (*e.g.*, VIP $\rightarrow$ RS IP translation). This design preserves functionality while eliminating vNIC limits caused by the vSwitch.

As shown in Fig. 10, the end-to-end path from tenant packet generation through the CStar Gateway to NF VM delivery now requires only one vNIC per NF VM to serve all tenant demands (compared to three in Fig. 3). Tenant-specific configurations are stored inside the NF VMs, indexed solely by the fake vNIC ID. The actions for NF access control, previously performed by the vSwitch, are also offloaded to the NF VM. The mapping from (VNI, Service IP) to fake vNIC ID is maintained centrally in the CStar Gateway. This design enables fine-grained multi-tenant and multi-demand NF access control without exhausting vSwitch vNICs.

Offloads Session Management. As shown in Fig. 11, CStar Gateway encapsulates the inner payload with a GENEVE header and adds a new outer VXLAN header with CStar VNI (system VNI). On the NF VM side, the vSwitch now records sessions using the CStar VNI (instead of each tenant's VNI) and the GENEVE-layer five-tuple (instead of the original inner five-tuple). Because all tenant flows destined for the same NF VM are tunneled through a single GENEVE endpoint with the same VNI, all concurrent flows handled by the NF VM collapse into one session entry in the vSwitch. This design eliminates session table exhaustion. As illustrated in Fig. 11, three distinct flows are treated as one by the vSwitch, consuming only a single session slot.

Prior to aggregation, distinct flows were mapped to separate session entries in the vSwitch, each carrying different actions (*e.g.*, pass, drop, meter) to perform per-tenant NF access control. After aggregation, the vSwitch applies a uniform action to all flows within the same GENEVE tunnel. To preserve fine-grained NF access control across multi-tenant, multi-demand traffic, some per-flow actions (*e.g.*, pass, drop, meter) must still be conducted inside the NF VM. As with vNIC aggregation, tenant-specific configurations that are used to derive session actions are migrated from the vSwitch into the NF VM. The NF VM reuses the vSwitch's original fast/slow path logic for session creation and action execution. The actions recorded in NF VM session entries form only a subset of those in the vSwitch (*e.g.*, excluding VXLAN decapsulation or vNIC steering). In this way, CStar Gateway shifts the entire session generation and storage burden from the vSwitch to the NF VM, enabling aggregation in the vSwitch and de-aggregation in the NF VM.

In the NF VM, the session entry key differs from that in the vSwitch as shown in Fig. 11. While the vSwitch uses (tenant VNI + inner five-tuple) to identify flows at the tenant and flow granularity, the original tenant VNI is replaced with the CStar system VNI during GENEVE encapsulation in the gateway, which makes it inaccessible to the NF VM. To preserve tenant-level isolation, we use the fake vNIC ID as the tenant identifier. Because the fake vNIC ID uniquely maps to a specific tenant and service demand, it serves as a functional substitute for

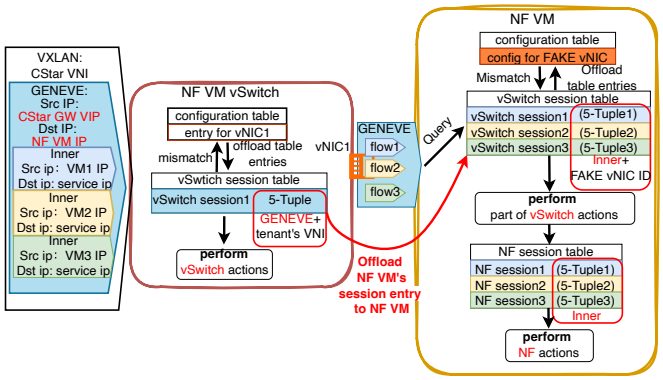

Figure 11: Aggregate session by GENEVE.

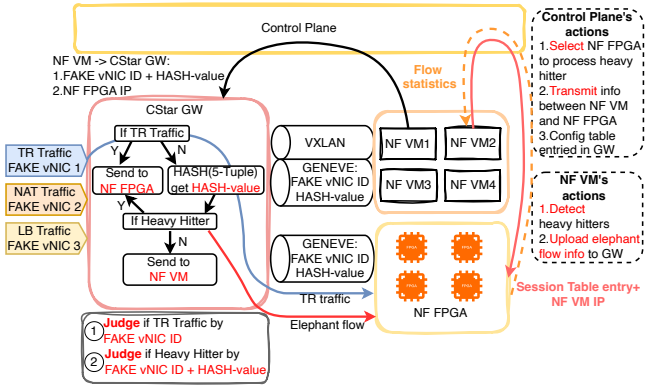

Figure 12: Offload TR and heavy hitter to FPGA.

the tenant VNI in session indexing. Thus, in the NF VM, the session key becomes (fake vNIC ID + inner five-tuple).

4.2 Handling TR Traffic and Elephant Flows with FPGA-Based NFs

Processing Elephant Flows with NF FPGA. To handle elephant flows, we deploy a pool of high-performance nodes. For cost-efficiency and customization, we chose to implement them using FPGAs, referred to as NF FPGAs. They process NF traffic at line rate, with programmable data-plane logic. However, because FPGA-based nodes have limited elasticity, NF FPGAs cannot scale rapidly to match dynamic NFV traffic demands. To address this, we adopt a hybrid architecture that combines NF FPGAs for elephant flows with elastic NF VMs for other workloads. Traffic is routed to the appropriate backend based on flow characteristics. This hybrid design enables the system to achieve both cloud-native elasticity and high-throughput elephant flow handling.

Since elephant flows cannot be predicted in advance, we perform real-time flow monitoring in NF VMs using per-flow packet and byte counters. As shown in Fig. 12, when a flow is detected as an elephant flow, the NF VM notifies the CStar Gateway through the control plane and provides the flow identifier. Upon receiving this signal, the CStar Gateway dynamically updates its forwarding policy so that subsequent packets of the elephant flow are redirected to an NF FPGA for high-throughput processing, while non-elephant flows continue to be routed to NF VMs. Elephant flows are identified by a composite key consisting of the fake vNIC ID and a hash value. The hash value, computed from the inner five-tuple, uniquely identifies a flow and is already calculated during the CStar Gateway ECMP process, which allows it to be embedded directly into the GENEVE header and passed to the NF VM as a stable flow identifier. The fake vNIC ID is prepended to prevent hash collisions across tenants.

To enable NF FPGAs to process elephant flows directly, the NF session entry containing the flow actions (*e.g.*, VIP-to-RS mapping) is migrated from the NF VM to the NF FPGA through the control plane after detection. Because elephant flows are identified mid-lifecycle rather than at the first packet, the NF VM already holds a valid session entry. Upon migration, the NF FPGA installs this entry and executes the same actions as the NF VM, ensuring functional equivalence and state continuity without disrupting flow processing.

For traffic analytics and billing, the NF FPGA collects per-flow statistics (*e.g.*, packet and byte counts) and reports them through the control plane to the NF VM that originally handled the flow. As shown in Fig. 12, the NF VM includes its own IP address when transferring the session entry to the NF FPGA to ensure correct delivery. The NF FPGA extracts the flow identifier from the session, pairs it with the NF VM's IP, and sends the annotated statistics through the control plane. These are then delivered to the target NF VM for statistic update.

Offloading I/O-intensive Traffic to FPGA. Although FPGA- and VM-based TR instances can achieve identical throughput, their computational resource allocation differs significantly. At the same line rate, VMs contain far more computational capacity (ops/sec) than required, whereas FPGA resources are concentrated on I/O processing. This I/O–optimized resource ratio makes FPGA a more efficient choice for TR workloads, which demand high throughput but minimal per-packet computation. In addition to improved resource alignment, NF FPGA incurs lower operational costs, directly enhancing the profit margin of TR services.

To steer TR traffic to NF FPGAs while keeping other NF traffic on NF VMs, we classify flows by service type. This is straightforward because tenants access NF services through assigned service IPs. The CStar Gateway deterministically identifies the traffic type using the tuple (Service IP, Tenant VNI). As shown in Fig. 12, this tuple maps to a fake vNIC ID that encodes tenant demand, including NF service type. The fake vNIC ID then enables the gateway to route TR-bound flows to NF FPGAs and all other flows to NF VMs.

4.3 Simplifying Elasticity via CStar Gateway.

To eliminate the need for consistent hashing in the vSwitch and its associated resource overhead and stability risks, we

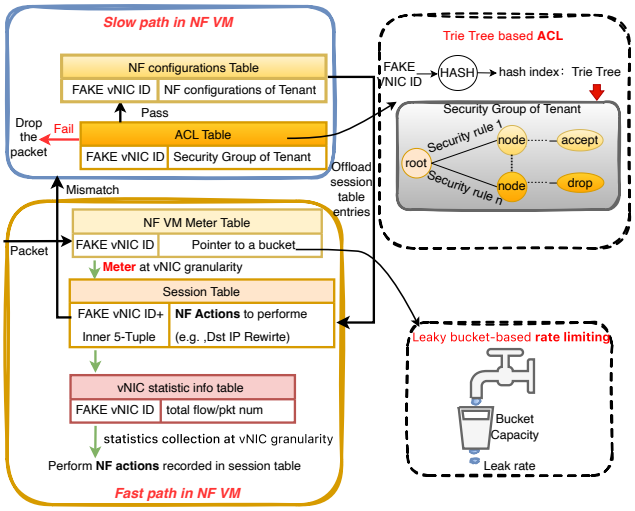

Figure 13: Implement ACL and Meter in NF VM.

migrate ECMP load balancing for NF traffic to the CStar Gateway. As shown in Fig. 10, tenant-side vSwitches now simply forward all NF traffic to the CStar Gateway with no ECMP required. Within the gateway, ECMP is performed by first resolving the target NF VM group using the flow's fake vNIC ID, which uniquely maps to a tenant demand, and then applying consistent hashing to select a specific NF VM and its host server. All of these processes are stateless.

The controller computes and maintains all consistent hash table entries used for ECMP. Without the CStar Gateway, scaling an NF VM group required the controller to push updates to hundreds of geographically distributed tenant-side vSwitches, which was slow and error-prone. With the ECMP logic centralized in the gateway, the controller updates only a single ECMP table in a few gateways, eliminating distributed reconfigurations. This architecture reduces control-plane load for NF VM scaling and shortens scaling latency.

5 System Implementation

5.1 ACL and Meter in NF VM

With CStar Gateway, tenant configs (ACLs, metering) move from vSwitch to NF VM, eliminating vSwitch dependency. NF VM now enforces full per-tenant-demand NF access control, not just NF service logic. When performing NF-service logic, we introduce a software-based fast/slow path separation packet processing pipeline within the NF VM, achieving near-line-rate performance through careful design. To achieve near-line-rate processing, we need to insert ACL and metering functions into this existing pipeline as illustrated in Fig.13.

ACL is inserted into the slow path, executing before any NF-specific logic. The steps are: 1) On the first packet, extract fake vNIC ID and retrieve associated security group rules; 2) Match inner 5-tuple against rules and drop immediately if denied; 3) Only if allowed, proceed to original NF slow-path logic and offload NF session entry to fast path. Placing ACL before NF slow-path logic ensures that only traffic permitted by security policies can access tenant NF configurations and trigger downstream processing. This effectively preserves the same vNIC-level NF traffic access control policies enforced by vSwitch and is now implemented entirely within the software data plane within NF VM. As shown in Fig.13, to maximize ACL matching efficiency, we organize each tenant's security group rules into a dedicated trie tree. The tree is indexed by fake vNIC ID, enabling O(k) prefix-based lookups per flow (k is the rule depth). To enforce vNIC-level rate limiting migrated from the vSwitch, NF VM firstly conducts metering in the fast path, before any further processing (ACL, NF logic). As shown in Fig.13, we adopt the leaky bucket algorithm, configured per fake vNIC ID with bucket capacity and drain rate. During initialization, NF VM pre-allocates a leaky bucket for each fake vNIC ID and builds a fast lookup table mapping fake vNIC ID $\rightarrow$ bucket pointer. After metering, NF VM can continue to process later processing logic (*e.g.*, look up session table, perform NF actions, get per-fake-vnic-id statistic).

Migrating per-vNIC configurations to the NF VM eliminates vSwitch pressure, but at the cost of additional NF VM memory. However, production telemetry shows that storing ACLs (as trie trees) and metering state (as leaky buckets) consumes only a few gigabytes per NF VM. Given that NF VMs are typically allocated tens of gigabytes of DPDK-managed hugepage memory, most of which remains unused. Memory exhaustion is not a practical concern, even under multi-tenant and high-rule-count workloads. NF VM forwarding operates in Run-To-Completion (RTC) mode, where each core independently executes the full data plane. Introducing ACL and metering adds measurable but bounded CPU overhead: 1) ACL (slow path): Applied per new flow, performs prefix matching via trie tree. This increases CPU utilization by 10–20% of baseline under high flow-setup rates (*e.g.*, 20% $\rightarrow$ 22–24%), but negligible under low rates. 2) Metering (fast path): Applied to all flows and adds a stable 10% baseline overhead due to per-packet leaky bucket updates. Since NF VMs were previously bottlenecked by vNIC/session limits instead of CPU, their cores typically ran at low utilization. Even with the added overhead, CPU usage remains well within safe operating margins, preserving headroom for scaling and burst handling.

5.2 Tofino based CStar Gateway

The CStar Gateway's data plane is inherently stateless and table-driven, involving only header parsing, table lookups, metadata tagging, and encapsulation. This is ideally suited for implementation on programmable switches like Tofino, which offer line-rate throughput and nanosecond-level per-packet processing. The core forwarding logic is decomposed into a sequence of match-action tables as shown in the Table 1: M/A table 1 is to get FAKE vNIC ID; M/A table 2 and 3 are to judge traffic type, if it is TR traffic or elephant flow, transmit

```
struct meta{
    bit<8>  group_id;  bit<32> flow_hash;   bit<10>  vm_id;       //consistent hash
    bit<32> nf_vm_ip;  bit<32> server_ip;
    bit<1>  encap_type;   //0: GENEVE, 1:VXLAN + GENEVE
}
register<bit<10>> consistent_hash_table {
    size = 262144;  // 256 * 1024, 256 NF-VM group, every group with 1024 slots
}
parser MyParser(packet, hdrs, meta){...}
control ingress(...){
    ...
        table fake_vnic_id_to_group_id{
        key={meta.fake_vnic_id:exact};
        actions(bit<8> VM_group_id){meta.group_id=VM_group_id;}
    }
    bit<18> flat_index = ((bit<18>)meta.group_id << 10) | (bit<18>)(meta.flow_hash%1024);
    bit<10> temp_vm_id;
    consistent_hash_table.read(temp_vm_id, flat_index);
    meta.vm_id=temp_vm_id;
    table nf_vm_id_to_nf_vm_ip_nf_server_ip{
        actions(...){meta.nf_vm_ip=temp_vm_ip; meta.server_ip=temp_server_ip;}
    }
    apply{
        ... // tables.apply()
        //encap geneve
        if (meta.encap_type==0){hdr.geneve.setValid(); hdr.vxlan.setInvalid();}
        //encap geneve and vxlan
        else {hdr.geneve.setValid(); hdr.vxlan.setValid();}
    }
}
```

Figure 14: P4 code to implement consistent hash in Tofino.

the flow to an NF FPGA IP (value of table 2 and 3); M/A table 4 and 5 are used to ECMP all other flow to a selected NF VM in a group using consistent hash. To implement Maglev consistent hashing in P4, we use a flat register array to store all NF VM group mappings. Each group has a dedicated hash table of 1024 slots, but instead of separate tables, we merge them into one large array. The index is computed as shown in Fig. 14, which maps each flow to a unique slot that stores NF VM information, enabling efficient ECMP lookup with minimal memory.

Table 1: Tables in Tofino: Entry Width and Entry Count.

M/A Table Name	Entry len(b)	Num
1.vni_service_to_fake_vnic_id	80	10W
2.TR_traffic_to_fpga_ip	56	3W
3.heavy_hitters_to_fpga_ip	88	128
4.fake_vnic_id_to_group_id	32	7W
5.nf_vm_id_to_ip_nf	74	1K

CStar Gateway is deployed at AZ granularity. We derive the entry width and total table size for each match-action table based on production-scale metrics, including total NF VM count, NF VM group count, tenant-service demand volume, TR tenant count, and peak concurrent elephant flows. Concrete values are summarized in the Table 1. Alibaba's Tofino switches feature four independent ingress/egress pipelines, each equipped with ~100 MB of SRAM and ~10 MB of DRAM. The CStar Gateway tables described above consume less than 20 MB total, occupying <20% of per-pipeline memory capacity. As a result, all four pipelines can operate in parallel under the current production scale. The design, with no pipeline folding required, fully utilizes the switch's maximum line-rate throughput, without memory-induced bottlenecks.

5.3 FPGA based NF

NF FPGA processes both TR traffic and elephant flows from all NFs. It must import fast-path session entries with their associated actions for elephant flows. Since different NF types define heterogeneous action sets (*e.g.*, NF1: action1, 2, 3; NF2: action2, 3, 4, 5), we design a unified session entry format. This format can express all types of action within a single, extensible table structure, enabling NF FPGA to execute any NF's logic without per-service specialization. As shown in Fig. 15, the session entry format in NF FPGA uses a composite key: (fake vNIC ID + inner 5-tuple), with a value field encoding the union of all possible actions across NF types. Each action consists of two parts: 1) Enable bit (1/0): indicates whether to execute the action; 2) Operand field: holds parameters required for execution (*e.g.*, IPv4/6 for SNAT/DNAT, port for rewriting). Aggregating all actions, a single session entry fits within 2 KB, which is sufficient to cover all NF types and their associated data. We implement NF FPGA on FM8 FPGAs, which provide hundreds of MB of on-chip BRAM and 4×4 GB of external DDR memory. To balance capacity, cost, and access latency, BRAM stores runtime metadata and temporary variables, while the large session table is placed in DDR.

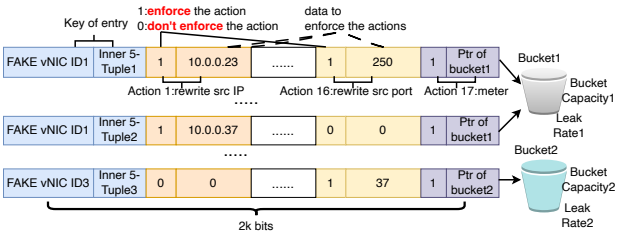

Figure 15: Fast-Path session entry format in NF FPGA.

To achieve higher per-instance bandwidth, we deploy four FPGA cards per server, forming a single logical NF FPGA instance. Each card delivers 200 Gbps of line-rate throughput, combining to an aggregate capacity of 800 Gbps for high-density NF processing. To process TR traffic, we adopt the hybrid execution: slow path (ACL + session install) runs on the host CPU and fast path (forwarding + metering) runs on the FPGA. Driven by unified 2KB session entries pushed from the host, the FPGA fast path ensures functional parity with NF VMs at line rate. Unlike NF VMs, where metering is implemented as a standalone fast-path stage, metering is embedded in session entry as a pointer to a shared leaky bucket (per fake vNIC ID) to enable tenant-granular rate limiting, as shown in Fig. 15. To process elephant with zero-disruption handoff, we migrated the session entries for elephant flows from NF VM to FPGA via the controller. Within the FPGA's fast path, beyond forwarding based on session entries, the FPGA also needs to collect per-fake-vNIC-ID flow stats (pkts/bytes) and report back to NF VM. These are used by cloud providers for billing and logging.

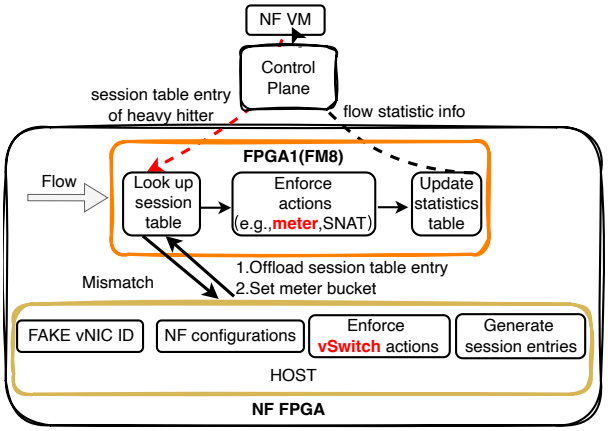

Figure 16: Processing pipeline of NF FPGA.

6 Evaluation

6.1 Experimental setting.

CStar Gateway and NF FPGA are deployed at AZ granularity. An NF FPGA instance (800 Gbps) is deployed to handle all TR and elephant flows within an AZ. Two identical Tofino-based CStar Gateway instances are deployed per AZ, with one serving as redundancy. The gateways share a single VIP, and the underlay network performs ECMP load balancing across both. Notably, this architecture has been running stably in Alibaba Cloud for nearly two years.

6.2 Experimental results.

The number of tenant NF demands served per NF VM can be determined by vNIC count (without gateway) or fake vNIC ID count (with gateway). As shown in Fig. 17, the number of tenant NF demands served per NF VM increased dramatically across all three AZs after deploying CStar Gateway. The improvement varies by AZ (11x, 11x, 5x). The variation reflects differences in NF VM type composition across AZs. We also observe that, by removing the vNIC quota bottleneck, a single VPC NAT NF VM can serve up to 7K tenant demands.

As shown in Fig. 18, we measure the concurrent flow count on a single L4-LB NF VM over 24 hours. Without CStar Gateway (blue curve), flow count plateaus at ~15M during peak hours (10:00–22:00), which is strictly capped by vSwitch session table limits. With CStar Gateway (orange curve), flow scale surges to 40M+. Instead of being constrained by the session table quota on the vSwitch, this number is influenced by the NF VM's own bandwidth, memory, and CPU capacity. This scaling gain is achieved without VM resizing or hardware upgrade, demonstrating CStar Gateway's ability to unlock latent capacity in existing NF VMs.

We measure the scaling latency from trigger (at 9 Gbps) to completion for expanding an NF VM group from 4 to 8 instances. As shown in Fig. 19, without CStar Gateway, the process takes ~11 seconds, due to distributed vSwitch state updates. With CStar Gateway, scaling completes in less than 2 seconds, achieving a 5× improvement. This scaling process involves only controller-to-gateway communication, eliminating vSwitch reconfiguration overhead.

We measure end-to-end latency from a tenant VM to a VPC NAT NF VM across packet sizes from 256B to 1500B, with CStar Gateway on the data path. As shown in Fig. 20, overall latency (blue curve) exceeds 290μs. The CStar Gateway's processing latency (orange curve) remains under 1.2μs, which is less than 0.4% of the total. These results show that CStar Gateway introduces negligible overhead, which can be transparent to tenant latency SLOs.

Fig. 21 compares fast-path per-packet latency (excluding first-packet setup) between VM-based and FPGA-based TR instances across packet sizes. FPGA processing (orange) is consistently $\geq$ 2.5 μs faster than VM (blue), demonstrating its higher data-plane efficiency.

Fig. 22 and Fig. 23 show one-week (8/22 22:00 – 8/29 22:00) traffic telemetry across three AZs. Despite elephant flow bursts pushing single-AZ FPGA traffic beyond 200 Gbps, our 800 Gbps-capable NF FPGA instances operate at <25% utilization, leaving ample space for future growth. Similarly, each CStar Gateway (multi-Tbps capacity) handles <250 Gbps peak (<10% of its limit), enabling further NF VM deployment without gateway scaling. TR traffic carried entirely by FPGA accounts for nearly 50% of all NF traffic volume.

Fig. 24 and Fig. 25 compare the daily throughput of two NF VM groups (8 NF VM VPC NAT group and 4 NF VM L4-LB group) without and with CStar Gateway. Without gateway (blue), both groups are severely throttled by vNIC/session limits. Specifically, the VPC NAT group is <0.7 Gbps. Despite 2× more VMs, its total throughput is lower than LB, with lightweight per-tenant flows. L4-LB group is $\leq$3.0 Gbps, constrained by session table exhaustion under high flow concurrency. With CStar Gateway (orange), the vNIC/session limits are removed. VPC NAT throughput increases fivefold to 3.5 Gbps without vNIC limits. L4-LB increases nearly sixfold to 17.5 Gbps without session limits. With the same VM count and specification, each VM now carries 5× more traffic.

As shown in Fig. 26 (I/O Utilization), all three AZs show a $\geq$5× increase in average NF VM I/O utilization after deploying the CStar Gateway. Without the gateway, NF VMs were underutilized due to vNIC and session limits. In Fig. 27 (CPU Utilization), CPU usage also rises by $\geq$5×, driven by increased per-VM traffic volume and the addition of in-VM ACL and metering logic in the data plane. Since vNIC-level configurations and session tables are offloaded to NF VMs, we also evaluate memory consumption. As shown in Fig. 28 (Memory Utilization), memory usage increases by less than 4.3% on average, indicating that this trade-off is efficient.

After the deployment of CStar Gateway, NF VMs are able to handle more traffic. As a result, fewer NF VMs are required for the same amount of workload, leading to a reduction in NF VM procurement cost. Table 2 shows the number of NF

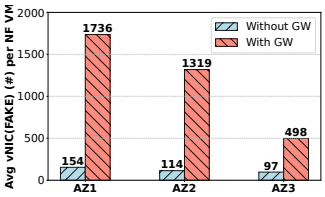

Figure 17: Avg num of FAKE vNIC/vNIC per NF VM with/without gateway (GW).

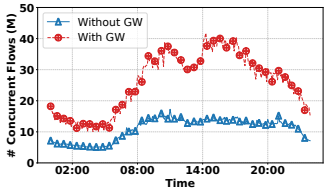

Figure 18: Concurrent flow per LB NF VM held with/without gateway.

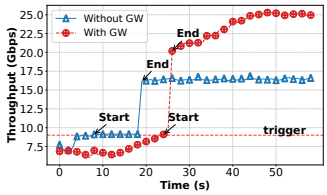

Figure 19: Elastic process with/without gateway.

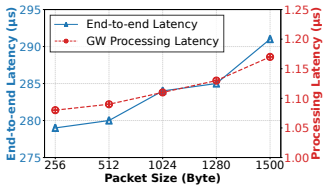

Figure 20: Latency of transmission and gateway processing under different packet sizes.

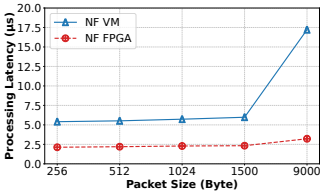

Figure 21: Processing latency of NF VM and NF FPGA.

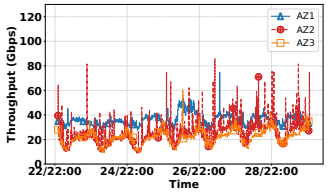

Figure 22: Throughput of NF FPGA in 3 AZs.

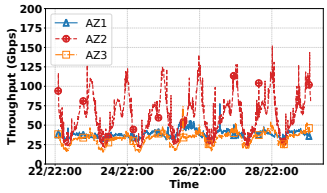

Figure 23: Throughput of CStar Gateway in 3 AZs.

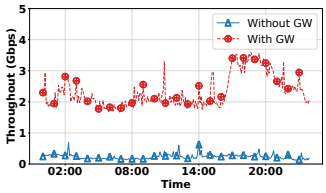

Figure 24: Throughput of one VPC NAT NF VM group with/without gateway.

Table 2: The number of NF VMs allocated in the AZs before and after the deployment of CStar Gateway.

	AZ1	AZ2	AZ3
Without CStar Gateway	92	124	127
With CStar Gateway	18	7	11

VMs required in AZs before and after deploying CStar Gateway. Even after considering the cost of the CStar Gateway itself, our expenses in the three AZs have reduced by 71.91%, 88.57%, and 85.69%, respectively.

7 Experience and Lessons

7.1 Single-Tenant I/O Starves Shared vNIC

After deployment, tenants occasionally reported NF service unreachability. Investigation revealed that during these incidents, over 99% of the aggregated vNIC bandwidth was consumed by traffic from a single fake vNIC ID, blocking flows from other tenants. The root cause was the location of rate control. While rate limiting was moved into NF VMs to enforce per-tenant SLAs and protect CPU/memory resources, NF VMs cannot control the arrival rate at the shared vNIC. Ingress rate control remained with the vSwitch, which lacked visibility into fake vNIC IDs and thus could not conduct tenant-granular limits. As a result, a single tenant could monopolize the shared vNIC's ingress bandwidth.

To mitigate this, we introduced coarse-grained rate limiting in the CStar Gateway. Traffic is first aggregated per NF VM, and if the total ingress rate exceeds a configured threshold, the gateway applies per–fake-vNIC-ID rate limiting for all flows mapped to that NF VM. This prevents any single tenant from saturating the shared vNIC and restores fairness at the entry point. However, the approach is not scalable. Each fake vNIC ID requires a dedicated leaky bucket, consuming scarce Tofino register memory. While feasible at the current scale (with spare capacity), accommodating larger tenant populations introduces scalability risks, motivating our future research on memory-efficient alternatives.

7.2 RSS Skew on vSwitch in GENEVE

After using CStar GW, wwwe observed a higher frequency of single-core hotspot events on vSwitches. Further analysis revealed that affected vSwitches almost always hosted one or more NF VMs. The root cause lies in how RSS is performed after VXLAN decapsulation. Without the gateway, the vSwitch applies RSS using the inner 5-tuple, which distributes flows evenly across cores. With the gateway, however, RSS is performed on the uniform GENEVE outer 5-tuple. As a result, flows are often mapped to the same vSwitch core, leading to single-core overload. To restore load balance, we randomized the GENEVE UDP source port on a per-flow basis. Since the UDP destination port must remain fixed for GENEVE demultiplexing, modifying the source port provides the necessary tuple diversity to spread flows evenly across vSwitch cores. This workaround requires no changes to vSwitch RSS logic or hardware, making it a practical and production-safe solution.

7.3 Why GENEVE?

We chose GENEVE over alternatives (*e.g.*, VXLAN-GPE, MPLS) for two key reasons. First, GENEVE natively supports extensible metadata through its TLV-based options, allowing us to embed flow-related context such as fake vNIC IDs or per-flow hash values for elephant-flow handling without redesigning the header. This extensibility also makes it easy to accommodate future needs by carrying additional control information in-band. Second, GENEVE was already the default encapsulation for NF-to-NF communication (e.g.,

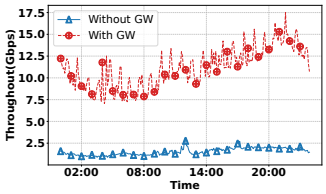

Figure 25: Throughput of one LB NF VM group with/without gateway (GW).

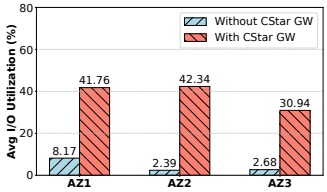

Figure 26: NF VM I/O utilization improvement with/without gateway.

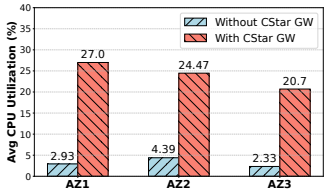

Figure 27: NF VM CPU utilization improvement with/without gateway.

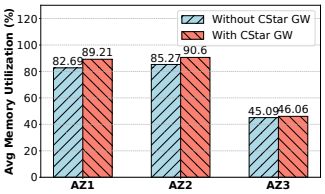

Figure 28: NF VM memory utilization improvement with/without gateway.

chaining VPC NAT with TR). As a result, NF VMs already support GENEVE. By adopting GENEVE in CStar Gateway, we avoided redundant development effort and established a unified metadata plane across the CStar Gateway, NF VM, and inter-NF paths, ensuring architectural consistency and long-term flexibility.

7.4 Metering as Separate Stage vs. in Session

Although metering is implemented in the fast path in both NF FPGA and NF VM, their realization is different. In NF FPGA, the metering is embedded as an action within each session entry. In NF VM, it is implemented as a standalone pipeline stage. This design choice was driven primarily by development effort. Given the severe resource mismatches faced by the CStar platform, we prioritized rapid functionality rollout. NF VMs deployed in production exhibit heterogeneous session entry formats; embedding the meter into entries would require per–NF VM adaptation. Implementing it as an independent pipeline stage allowed code reuse across multiple DPDK-based NF VMs, significantly reducing engineering overhead. By contrast, Alibaba's FPGA-based fast paths already included this design pattern, where meters were attached to session entries as actions. Reusing this pattern enabled rapid integration while minimizing development cost.

7.5 Why We Use (fake vNIC ID, hash value) not 5-Tuple to Identify Elephant Flows

We identify each elephant flow using a combination of the flow's hash value and its fake vNIC ID. The hash value is computed once at the CStar Gateway and then conveyed to NF VMs through GENEVE. We avoid recomputing the hash at NF VMs, since their existing hash functions differ from that of the gateway. Enforcing consistency would require intrusive modifications across NF VMs. We also avoid using the full 5-tuple as the identifier. First, NF VMs need to report the identifier back to the gateway via the control plane, and transmitting a compact hash value has lower overhead than carrying a full 5-tuple. Second, matching 5-tuples would require the gateway to parse inner packet headers and maintain larger match tables, consuming more hardware resources. For these reasons, we use both the hash value and fake-vNIC-ID as the practical and resource-efficient identifier for elephant flows.

8 Related work

Compared to hardware middleboxes, NFV offers service flexibility, cost efficiency, and resource optimization for public cloud vendors. However, achieving line-rate packet processing and horizontal scalability [10,12,20,21,23,26] pose significant challenges for NFV implementations. Azure [4] initially enhanced single-node processing by deploying SmartNICs on servers [8] within their VPC infrastructure. However, this approach reduced NFV flexibility and led to resource underutilization. To address this, they adopted a shared resource pool design, which reduced the per-server processing capability and redirected overflow traffic to the pools. Google's Andromeda [5] implemented complex NFV functionality entirely in software VMs. However, its architecture ensures that each tenant is allocated a dedicated VM to execute network functions. They do not support the scenario where multiple tenants share the same NF VM. While prior NFV systems scale by adding hardware or VMs, our work tackles this resource imbalance using existing NF VMs. We offload vSwitch's overhead (vNICs, sessions) into underutilized NF VMs, and selectively accelerate heavy flows (e.g., elephant flows, TR) via FPGA to preserve elasticity and performance. Prior NFV systems generally scale by adding hardware or VMs. In contrast, our work addresses resource imbalance within existing NF VMs. We offload vSwitch's overhead (vNICs, sessions) into underutilized NF VMs, while selectively accelerating heavy flows using the FPGA. This approach preserves both elasticity and performance without requiring additional hardware or VM.

9 Conclusion

In this work, we introduce a design combined with the Tofino-based gateway and FPGA to solve resource mismatch problems when building scalable NFV products using standard VMs. This solution significantly reduces the cost of deploying NFV systems on public clouds based on standard VMs, without requiring intrusive changes to the underlying cloud virtualization architecture. It offers a viable path for other cloud service providers seeking to deliver cost-effective and cloud-based NFV services.

References

[1] Alibaba cloud transit router. https://www.alibabacloud.com/en/product/transit_router, 2025.

[2] Features and metric data of ecs instance families. https://www.alibabacloud.com/help/en/ecs/user-guide/overview-of-instance-families, 2025.

[3] Deepak Bansal, Gerald DeGrace, Rishabh Tewari, Michal Zygmunt, James Grantham, Silvano Gai, Mario Baldi, Krishna Doddapaneni, Arun Selvarajan, Arunkumar Arumugam, et al. Disaggregating stateful network functions. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1469–1487, 2023.

[4] Deepak Bansal, Gerald DeGrace, Rishabh Tewari, Michal Zygmunt, James Grantham, Silvano Gai, Mario Baldi, Krishna Doddapaneni, Arun Selvarajan, Arunkumar Arumugam, et al. Disaggregating stateful network functions. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1469–1487, 2023.

[5] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, et al. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX symposium on networked systems design and implementation (NSDI 18)*, pages 373–387, 2018.

[6] 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.

[7] 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 17)*, pages 315–328, Boston, MA, March 2017. USENIX Association.

[8] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, April 2018. USENIX Association.

[9] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review*, 44(4):27–38, 2014.

[10] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 44(4):163–174, 2014.

[11] J Gross, I Ganga, and T Sridhar. Rfc 8926: Geneve: Generic network virtualization encapsulation, 2020.

[12] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. Stateless network functions: Breaking the tight coupling of state and processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 97–112, Boston, MA, March 2017. USENIX Association.

[13] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, 1997.

[14] Praveen Kumar, Nandita Dukkipati, Nathan Lewis, Yi Cui, Yaogong Wang, Chonggang Li, Valas Valancius, Jake Adriaens, Steve Gribble, Nate Foster, and Amin Vahdat. Picnic: predictable virtualized nic. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 351–366. 2019.

[15] Jianyuan Lu, Tian Pan, Shan He, Mao Miao, Guangzhe Zhou, Yining Qi, Biao Lyu, and Shunmin Zhu. A two-stage heavy hitter detection system based on cpu spikes at cloud-scale gateways. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 348–358. IEEE, 2021.

[16] Colm MacCarthaigh. Multi-tier stateful network flow management architecture, June 12 2018. US Patent 9,998,955.

[17] Colm MacCárthaigh. Workload isolation using shuffle-sharding. *Tech. Rep.*, 2019.

[18] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[19] 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.

[20] 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. *ACM SIGCOMM Computer Communication Review*, 43(4):207–218, 2013.

[21] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/Merge: System support for elastic execution in virtual middleboxes. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 227–240, Lombard, IL, April 2013. USENIX Association.

[22] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K Sitaraman. Akamai dns: Providing authoritative answers to the world's queries. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 465–478, 2020.

[23] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. Rollback-recovery for middleboxes. volume 45, page 227–240, New York, NY, USA, August 2015. Association for Computing Machinery.

[24] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *SIGCOMM Comput. Commun. Rev.*, 42(4):13–24, August 2012.

[25] Enge Song, Nianbing Yu, Tian Pan, Qian Fu, Liang Xu, Xionglie Wei, Yisong Qiao, Jianyuan Lu, Yijian Dong, Mingxu Xie, Jun He, Jinkui Mao, Zheng-Jun Luo, Chenhao Jia, Jiao Zhang, Tao Huang, Biao Lyu, and Shunmin Zhu. MIMIC: SmartNIC-aided Flow Backpressure for CPU Overloading Protection in Multi-Tenant Clouds. pages 1–11, 2022.

[26] Shinae Woo, Justine Sherry, Sangjin Han, Sue Moon, Sylvia Ratnasamy, and Scott Shenker. Elastic scaling of stateful network functions. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 299–312, 2018.

[27] Tingting Xu, Bengbeng Xue, Yang Song, Xiaomin Wu, Xiaoxin Peng, Yilong Lyu, Xiaoliang Wang, Chen Tian, Baoliu Ye, Camtu Nguyen, Biao Lyu, Rong Wen, Zhigang Zong, and Shunmin Zhu. CyberStar: Simple, elastic and Cost-Effective network functions management in cloud network at scale. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 227–246, Santa Clara, CA, July 2024. USENIX Association.

[28] 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, Renton, WA, April 2022. USENIX Association.

Appendices

A Additional Experiences

A.1 CStar Gateway Is Stateless

A core design principle of CStar Gateway is keeping the gateway stateless: unlike NF VMs, which maintain per-flow session state, the gateway performs only table-based, metadata-driven forwarding. It stores no flow state, maintains no connection tables, and has no concept of fast/slow path. By being stateless, we can deploy multiple identical gateway instances per AZ, fronted by a single VIP and load-balanced via ECMP. This architecture delivers three critical production benefits: (1) horizontal scaling under traffic surge, (2) seamless failover during hardware/software faults, and (3) safe, incremental upgrades via gray deployment, all without service disruption.

A.2 Method to handle elephant flow before using FPGA

Before using CStar Gateway and NF FPGA, our elephant flow mitigation relied entirely on NF VMs and vSwitches: cores were monitored for saturation; top flows by packet rate were identified; and tenant-side vSwitches were instructed to rate-limit those flows(backpressure) to relieve CPU load. While functional in lab environments, this approach proved unacceptable in production: the sudden rate limiting at vSwitch caused bursty packet drops, violating tenant SLAs and destabilizing latency-sensitive applications. This forced a fundamental redesign: we offloaded elephant flow handling to NF-FPGA to eliminate packet loss during throttling, ensuring SLA compliance even under Gbps-scale bursts.

A.3 Why not implement consistent hash in vSwitch

Consistent hashing is typically deployed in overlay (not underlay) due to hardware constraints, but we do not implement it in vSwitch, even for NF-VM scaling. The reasons are:(1) maintaining per-group hash tables offers no benefits for general tenant VMs and no other workload except NF traffic requires it. Making dedicated consistent hashing support would cause a waste of memory and control-plane capacity. (2) vSwitch is a foundational multi-tenant component. Adding consistent hashing to vSwitches increases the probability of failure. Since vSwitch failures inherently impact a wide range of services, any failure can result in severe, widespread problems.

A.4 Optimizing Tofino Memory Usage

Implementing gateways on Tofino faces one hard constraint: on-chip memory (SRAM, TCAM, Register) is finite and precious. While our deployment hasn't hit limits, we designed CStar Gateway with memory efficiency as a first-class goal. A representative optimization is that our consistent hash table (backed by register arrays) stores only an NF VM identifier (10 bits per slot) and not the full 64 bit (NF VM IP, server IP) tuple. Because different slots frequently map to the same backend VM, this indirection reduces memory consumption. A small, static secondary table then maps VM ID to network addresses, which adds minimal latency while enabling far greater scale.

A.5 Multi-NF Service Chaining with no addictive vNIC use

As tenant network requirements grow increasingly sophisticated, it is no longer feasible to satisfy all needs with a single NF. Instead, multiple NFs must be chained together to deliver end-to-end connectivity while preserving isolation, security, and performance. A representative example is a tenant's access to Alibaba Cloud's Simple Log Service (SLS). It is a centralized, shared service hosted in a dedicated VPC. To enable conflict-free communication, we deploy VPC NAT Gateways, which perform source/destination IP translation to avoid routing ambiguity. However, VPC NAT alone is not sufficient. NAT only rewrites IP headers; it does not perform routing decisions between VPCs. The actual L3 forwarding across VPCs is handled by the TR. Thus, the complete data path for a tenant accessing SLS becomes: Tenant VM $\rightarrow$ VPC NAT $\rightarrow$ TR $\rightarrow$ SLS VPC VM. A key insight in our design is that vNICs need only be allocated at the entry point of the NF chain, not at every hop. When traffic first enters the chain (at VPC NAT NF VM), it is assigned a dedicated vNIC that is bound to the tenant demand's specific NF configuration. This ensures that NF policies tailored to each tenant's demands are enforced at the entry point of the NF chain. After processing by VPC NAT, the packet is encapsulated with metadata (e.g., fake vnic id) and forwarded to TR. TR does not require its own per-tenant-demand vNIC because the traffic has already been classified by VPC NAT. The only exception is when two tenant VPCs are directly peered with non-overlapping CIDRs and traffic flows Tenant VPC $\rightarrow$ TR $\rightarrow$ Peer VPC without NAT. In this scenario, TR must use per-tenant vNICs to enforce isolation since no upstream NF has performed classification.

A.6 How NF FPGA Achieves Elasticity for TR Traffic?

Although NF FPGA lacks the elastic agility of VMs, we deploy it for TR workloads because VM-based scaling proves economically unsustainable due to I/O intensity. To mitigate FPGA's elasticity limitation, we implement a two-pronged strategy: (1) Each NF FPGA instance supports up to 800 Gbps which exceeds current AZ-wide TR demand and absorbs short-term bursts without scaling; (2) We employ predic-

tive traffic analytics and a tunable scaling threshold to trigger instance provisioning before capacity exhaustion which ensures zero packet loss or latency spike during scale events. This design mitigates FPGA's inability to handle elastic TR traffic by introducing limited elasticity, narrowing the gap between performance and elasticity.