CyberStar: Simple, Elastic and Cost-Effective Network Functions Management in Cloud Network at Scale

Tingting Xu, *Nanjing University;* Bengbeng Xue, Yang Song, Xiaomin Wu, Xiaoxin Peng, and Yilong Lyu, *Alibaba Group;* Xiaoliang Wang, Chen Tian, Baoliu Ye, and Camtu Nguyen, *Nanjing University;* Biao Lyu and Rong Wen, *Alibaba Group;* Zhigang Zong, *Alibaba Group and Zhejiang University;* Shunmin Zhu, *Alibaba Group and Tsinghua University*

https://www.usenix.org/conference/atc24/presentation/xu-tingting

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

CyberStar: Simple, Elastic and Cost-Effective Network Functions Management in Cloud Network at Scale

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[‡¶*]
[†]*Nanjing University* [‡]*Alibaba Group* [§]*Zhejiang University* [¶]*Tsinghua University*

Abstract

Network functions (NFs) facilitate network operations and have become a critical service offered by cloud providers. One of the key challenges is how to meet the elastic requirements of massive traffic and diverse NF requests of tenants. This paper identifies the opportunity by leveraging cloud elastic compute services (ECS), i.e. containers or virtual machines, to provide the cloud-scale network function services, CyberStar. CyberStar introduces two key designs: (i) resource pooling based on a newly proposed three-tier architecture for scalable network functions; and (ii) on-demand resource assignment while maintaining high resource utilization in terms of both tenant demands and operation cost. Compared to the traditional NFs constructed over bare-metal servers, CyberStar can achieve 100Gbps bandwidth (6.7×) and scale to millions of connections within one second (20×).

1 Introduction

Cloud network offers customers not only transportation but rich network functions (NFs), e.g., network address translation (NAT), load balancers (LB), firewall (FW) [25, 41, 54, 61, 65, 66, 68]. Conventionally, these network functions are deployed on bare-metal server clusters to maintain good performance [45, 63, 74]. However, with the rapid growth of enterprises that migrate their business to the cloud [27, 49, 68, 70], the bare-metal server-based network functions cannot meet the flexible scalability demand. For example, during the period of shopping festivals (e.g., Double-11 [3]) or live broadcasting, the traffic can increase by 100 times or even 1000 times in a very short time, which needs a large number of NFs, like load balancers, NATs, etc. Therefore, in practice, cloud network operators must reserve a large number of bare-metal servers for emergency events. To reduce the high operational costs and long setup times associated with this approach, cloud networking service providers are seeking elastic solutions that can dynamically respond to changing business demands.

[◇]Co-first authors [*]Co-corresponding authors

A natural solution for the network function service is leveraging the cloud-native elastic compute/container services (ECS), e.g., virtual machines (VMs), containers [2, 6, 7]. This approach offers multiple benefits for NFs deployment: (i) *"infinite" computation resource*. It allows the NF platform to scale using the vast and virtually limitless resources provided by cloud service providers. We can apply the "infinite" ECS resource which mitigates the impact of the long setup time of bare-metal servers. (ii) *"pay-as-you-go" price model*. Cloud ECS eliminates the need for users to pay for over-provisioned resources, resulting in significant cost savings. (iii) *high availability* against infrastructure failure, planned downtime and software upgrades to minimize the impact on customers.

Therefore, we aim to develop an elastic cloud-native NF management platform over ECSs. Elasticity in this context means that the system's capacity can continuously align with real-time load fluctuations. Building such an elastic NF platform involves pursuing several key objectives: rapid scalability, high resource utilization, and easy management. (i) Scalability of the NF platform is the system's ability to rapidly scale up to handle millions of simultaneous connections for individual tenants and scale out to serve cloud-scale tenants. Notice that, the scalability of NFs is not solely confined by ECS scalability but by its intricate internal execution logic and state consistency requirement. (ii) Maintaining high resource utilization is crucial for operators to reduce their operational costs. However, it is challenging because the traffic distribution among tenants demonstrates considerable variation and difference. (iii) Ease of management is crucial for the NF platform. In the diverse realm of cloud network resources, the multitude of ECS configuration options and the inherent delays and constraints in resource allocation pose significant challenges in ECS selection, requiring meticulous consideration of performance, cost and availability.

We propose CyberStar, a three-tier NFs management platform that leverages ECS to achieve high elasticity, cost-effectiveness and flexibility. We notice it is not a good option to directly apply the architectures designed for individual NFs, such as monolithic NF instances or two-tier architectures [10, 44, 70]. Installing a monolithic VNF in one ECS instance suffers from shortcomings like high configuration complexity of ECSs, hard-to-realize hardware optimization, low resource utilization across different NFs and variable traffic. Two-tier architectures proposed by pioneer elastic scaling works [44, 70] extract the state from the NFs to achieve free scalability of the stateless components. However, the stateful components cannot scale out well and states maintained remotely introduce non-negligible overhead, including latency inflation, extra CPU cycles, and bandwidth consumption during the access of the remote state.

By investigating the operations of various NFs and the corresponding optimization approaches [25, 47, 57, 72], we observe that different NFs follow the same match-action pipeline structure in the data path, even though they implement different match-action operations. Therefore, we decompose the packet processing pipeline into multiple NF-independent match-action units, allowing the assembly of these units to various NF types (§4.2). This design offers several benefits. (i) High scalability: the match-action unit only caches rules that indicate how to process packets, eliminating the necessity of maintaining state consistency. (ii) High utilization: the match-action is NF-independent, so it can be shared by different NFs, leading to high resource utilization. (iii) High performance: the design of function independence avoids redundant deployment and optimization efforts on packet processing across different NFs. It also facilitates processing acceleration using heterogeneous hardware resources in the cloud.

After determining the NF-independent packet processing, the remaining NF-specific parts are placed in the service computation (SC) plane, which is responsible for computing rules based on NF-specific state and logic. However, such stateful components hinder scalability. We observe that the NF-specific state can be partitioned at a fine granularity, allowing SC to scale effectively when deployed across numerous instances. Based on this observation, we design a scaling-in and scaling-out mechanism for the SC plane, eliminating traffic halting due to waiting for state synchronization (§4.1).

Both the PP and SC planes consist of numerous lightweight instances deployed on ECSs. However, the connections between PP and SC nodes are still tightly coupled. Each PP instance must record the information of requests, NF types, and the location of corresponding SC instances. When the SC plane scales (i.e., membership changes), this information must be notified to the relevant PP nodes to ensure the correct routing of requests. Similarly, when the PP plane scales, any changes to PP nodes must be communicated to the relevant SC nodes. To decouple the scaling, we introduce the Fabric Master (FM), which manages the connections between the SC and PP planes (§5). The FM shields the scaling of the PP and SC plane from each other by taking over the responsibility of delivering requests to the appropriate SC instances.

To this end, CyberStar introduces the architecture with loosely coupled components to support network functions

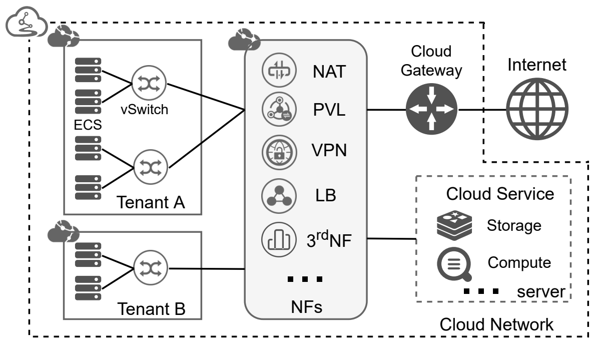


Figure 1: NFVs in cloud networks

including packet processing, fabric master, and service computation (§4 and §5). Built on the three-tier architecture, CyberStar facilitates the management of heterogeneous hardware resources in the cloud to accelerate packet processing (§6). CyberStar achieves high resource utilization by sharing resources across tenants, network functions, and traffic, and employs an auto-scaling mechanism based on Deep Reinforcement Learning (DRL) for optimal long-term resource use (§7.1). For reliability, CyberStar uses Shuffle Sharding [67] and local rate management to avoid the impact of shared malicious nodes (§7.2).

We deploy CyberStar in a cloud-scale production network for years, verifying its capability to meet the diverse demands of tenants and handle high traffic volumes. Compared to traditional NFs deployed in dedicated zones, CyberStar demonstrates rapid scalability to serve cloud-scale tenants and traffic. In terms of performance, CyberStar maintains a throughput of 26 packets per second ($\sim 6.7\times$) compared to native ECS-based solutions, and can process millions of connections per second ($20\times$) for a single tenant instance. To the best of our knowledge, this work is the first to systemically address the challenges and opportunities to build a network functions management system through cloud ECS service. Though CyberStar is designed for network functions management, its insights and designs are valuable for building other cloud-native applications as well.

2 Background and Motivation

2.1 Background

Figure 1 shows the panorama of network services in the cloud. In addition to the basic communication service, cloud networks provide advanced network functions, such as Network Address Translation (NAT), Private Link (PVL), Virtual Private Network (VPN), Load Balancer (LB), etc. These network functions play an important role for the connections among (1) Tenant Virtual Private Clouds (VPCs); (2) Customers' VPCs and their on-premise data centers; and (3) Internet users and Cloud service. For example, a private link bridges a tenant VPC to cloud computation and storage service with high bandwidth and secure connection. For Cloud-to-Internet

NFs	Capability
NAT	2 million connections; 100 thousand CPS;
LB	100 million connections; 1 million CPS; 100 thousand QPS
IPSec VPN	5∼ 200 Mbps bandwidth

Table 1: Requests of one tenant in the cloud. (CPS: Connections Per Second, QPS: Queries Per Second)

communication, the Source NAT (SNAT) offers the ability to access the Internet for virtual machines (VMs). The Destination NAT (DNAT) allows the traffic from the Internet to reach VPC. Specifically, customers can also define and place their own NFs (3rd NF) in the cloud.

With the increasing migration of businesses to the cloud, the demand for robust network solutions has surged [1, 4, 14]. These network functions need to be rapidly provided and iterated upon by cloud vendors to meet evolving requirements. Traditional, closed, and standard hardware middleboxes cannot keep pace with these dynamic cloud network demands due to the long development cycle, limited customization and lack of programmability. Cloud vendors are widely adopting Virtualized Network Function (VNF) technology. VNF employs general-purpose x86 servers to build network functions, which greatly improves flexibility, manageability, and cost-efficiency [9, 27, 33, 55]. It has shown that the software middleboxes are able to offer equivalent functionalities as the corresponding hardware implementations [29, 39]. The advance of software-defined networking (SDN) further facilitates the deployment of the software middleboxes running in the cloud [27, 33, 41, 68].

However, the typical online cycle of x86 servers is inadequate for addressing unpredictable business demands. The conventional process of constructing new bare-metal clusters, which involves purchasing, constructing, configuring and verifying, can take months. This lengthy preparation period poses challenges, especially during critical periods like shopping festivals, where there are demands for advanced planning. For unpredictable events without advanced plans, this approach proves to be inadequate. Cloud service providers can establish large resource pools to handle peak demands for business emergencies. However, the cost of maintaining such a large amount of infrastructure remains a significant challenge. Therefore, relying on a fixed bare-metal resource pool is not a sustainable long-term solution.

2.2 Motivation and Challenges

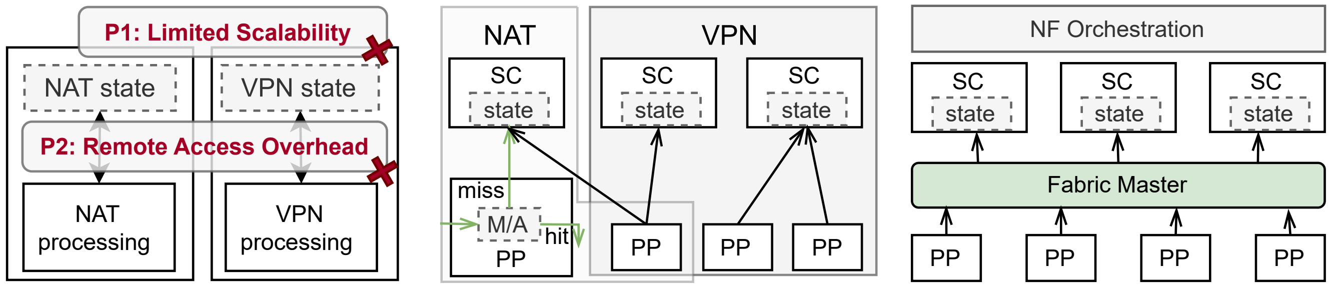
Cloud-native technologies significantly simplify both development and operations for tenants. A straightforward approach to building the NFs platform involves replacing bare-metal servers with cloud ECS resources. This transformation presents both opportunities and challenges. We aim to build a general NFs platform over ECS in practice, achieving the following objects:

Elastic Scalability. Achieving elastic scalability requires the NF platform to seamlessly and rapidly adapt to both increasing and fluctuating demands. To accommodate growing demands, the platform must scale out to serve cloud-scale tenants and scale up to handle millions of simultaneous connections for individual tenants. For instance, as illustrated in Table 1, a tenant may require millions of simultaneous connections for NAT and generate 1 million new connections per second for LB. Responding rapidly to dynamic demands involves more than just resource allocation. NF scalability is influenced not only by resource availability but also by complex internal execution logic and state consistency requirements. These constraints become more apparent when addressing fluctuating workloads. Therefore, the key to enhancing NF scalability lies in strategically decoupling the packet processing from state management. This entails ensuring that each NF component deployed in an ECS is independent and capable of scaling on demand.

High Resource Utilization. The NF platform serves cloud-scale users, necessitating the utilization of a substantial number of ECS resources. However, the low utilization of cloud ECS has become a key problem of today's cloud providers [5,11,24,26,31,37,52,61,77]. The average CPU utilization of 60% VMs in Azure is less than 20% [24]. The average CPU utilization of Alibaba cluster is between 20% to 50% in the majority of times [37]. Keeping long-term efficient resource utilization is critical in effectively reducing operational costs.

The challenge in improving resource utilization lies in the significant variations of traffic distribution among different users, as well as the large fluctuations in traffic for individual users. In the dimension of tenants, one tenant may have modest traffic of 30 Mbps distributed across 300K routes, while another tenant experiences substantial traffic of 200 Gbps on just 7 routes [4]. If we distribute a tenant's traffic across multiple ECS instances, in the former scenario, a single ECS instance is sufficient to support the traffic. However, in the latter scenario, multiple instances would be required to handle the tenant's traffic, while each ECS instance needs to manage an average of 30 Gbps with regard to 7 instances. In the time dimension, instance provisioning typically aligns with peak bandwidth usage. However, we have observed that peak-to-average bandwidth demands can fluctuate significantly, sometimes by up to a factor of 100. This variation poses a challenge in efficiently provisioning instance capacity. Therefore, a fine-grained scheduling approach is necessary to process tenant workloads in a shared manner, reducing the overall peak-to-average ratio of the load.

Low Management Complexity. Scaling an NF platform to serve cloud-scale tenants involves the effective management of numerous instances with diverse resource configurations. Regarding the diverse realm of cloud resources, the multitude of configuration options can actually pose challenges in ECS selection, cost, and availability. It requires cloud providers to meticulously consider and balance multiple factors since it is hard to estimate the resource consumption of NFs belonging



(a) Typical NF Architecture (b) 1st Generation of CyberStar (c) 2nd Generation of CyberStar

Figure 2: NF architectures. (a) Typical NF architecture. Prior efforts [44, 70] extracted state from NF instances to improve the scalability of packet processing. (b) To improve the scalability of stateful components and reduce the overhead of state fetching, the 1st generation of CyberStar puts the state into a distributed service computing (SC) plane and adapts network-independent packet processing (PP). (c) The 2nd generation of CyberStar introduces a fabric master, decoupling the packet processing and service computing.

to a specific tenant. Moreover, the process of creating new resources on large public clouds introduces inherent delays and constraints in resource allocation. For example, provisioning virtual machines that surpass vCPU quotas may entail waiting times ranging from several hours to a day. To simplify management, the straightforward approach is to apply *a few types of prevalent and low-configured* ECS instances in clouds.

Reliability and Transparency. A general cloud NFs platform must seamlessly avoid interruptions caused by customer workloads affecting each other. This shared platform demands high reliability to ensure isolation, preventing the malicious traffic of one tenant from impacting another. Additionally, the implementation details and placement of network functions in the cloud environment must be transparent to tenants. This transparency does not conflict with customization. Users can still customize their NF services through the orchestration of NF chains, tailoring the platform to meet their requirements without needing to know the implementation details.

3 CyberStar

3.1 Design Rationale

Elastic Scalability using Disaggregated Architecture. The pioneer elastic scaling works of network functions [10,44,70] extracted the state from the NFs to achieve free scalability of the stateless components, as shown in Figure 2a. However, we cannot directly apply the two-tier architectures for two reasons: (i) The stateful component itself cannot scale out well under the cloud-scale traffic. (ii) States maintained remotely introduce non-negligible overhead, including increased latency, extra CPU cycles and higher bandwidth consumption during the access of the remote components.

Addressing the scalability for cloud-scale traffic and multiple tenants necessitates the partitioning of NF state and operation into lightweight components and distributing them across the massive ECS instances. By investigating the architecture of various NFs and the corresponding optimization approaches [25, 47, 57, 72], we observe that NF execution can be separated into NF-independent packet processing, NF-specific logic operation, and NF-specific state. Due to the reliance on NF-specific logic execution on NF state, they should be placed together for fast state access.

To this end, the first generation of CyberStar is proposed, whose architecture is shown in Figure 2b. It partitions the NFs into two components: high-speed packet processing (PP) and service computations (SC). The PP plane is NF-independent, focusing on providing high-speed packet processing. The SC plane is NF-specific, performing service computations based on NF-specific state and logic, and generating rules that direct PP on how to process packets. The stateless PP plane only caches rules rather than maintaining state. The NF-specific state is locally maintained by the SC plane for fast access, but this hinders scalability. We observe that NF-specific states can be partitioned at a fine granularity, allowing SC to be divided into numerous instances. Based on state partitioning, the state on different SC instances does not need to maintain constant consistency. State synchronization of the SC plane is only triggered during scaling events, and the process does not involve halting traffic.

The connections between PP and SC nodes affect the scalability of PP and SC planes. Each PP instance must identify the tenant requests, type of NF and addresses of SC instances. When the SC plane scales (i.e., membership changes), this alteration must be notified to the corresponding PP nodes so that they can route requests correctly. Similarly, when the PP plane scales, any changes to PP nodes must be communicated to the relevant SC nodes. To decouple the scaling, we introduce the Fabric Master (FM) in the 2nd generation of CyberStar as shown in Figure 2c. The Fabric Master manages the connections between the SC and PP planes, taking responsibility for dispatching requests from the PP plane to the SC plane, thereby shielding the effects of membership changes on SC and PP during scaling.

Function-Independent Packet Processing. Different NFs may implement various match-action operations in their data paths, but they typically adhere to the same pipeline match-action structure. Our key insight is that we can place a parser-match-action unit to an ECS, and the packet processing of a dedicated NF can be realized by orchestrating those match-action units [13, 15]. Specifically, since the match-action unit is NF-independent, it can be shared by multiple NFs and multiple tenants. As shown in Figure 2b, VPN and NAT can share the same action of revising the packet header. We can further manage the pipeline of these units to fulfill: i) NFs chain, e.g., FW-NAT; ii) complex NFs e.g., IPSec VPN.

Furthermore, to stay at the cutting edge, modern cloud infrastructure adopts heterogeneous devices, including ECS based on Intel and ARM architecture, DPUs/SmartNICs, and other advanced hardware accelerators. Based on the modular packet processing design, we can leverage the widely deployed DPU/SmartNIC devices to accelerate packet processing efficiently.

High Utilization with Resource Sharing and Auto-Scaling. Achieving high resource utilization is crucial for service providers to reduce costs, especially in the context of cloud-scale workloads and increasing NF instances [5, 11, 24, 26, 31, 37, 52, 61, 77]. With regard to the disaggregated design of the NFs platform, we can achieve high resource utilization by effectively sharing the resource in three dimensions: Tenants, Network Functions, and Traffic. However, conventional manual management of resource scaling can be less effective given the scale of cloud workloads. CyberStar employs an auto-scaling mechanism with a global view, based on Deep Reinforcement Learning (DRL), to optimize long-term resource utilization.

Reliability based on Shuffle Sharding and Rate Management. CyberStar needs to minimize the impact of malicious traffic to improve reliability. On the one hand, CyberStar adopts *Shuffle sharding* [67], an effective method for segregating tenants' workloads by distributing traffic across multiple instances with minimal overlap. On the other hand, CyberStar incorporates local rate management at each instance to ensure fair sharing and work conservation among tenants.

3.2 Architecture

Figure 2c illustrates the 2nd generation architecture of CyberStar, which consists of an NF Orchestration and three-plane network functions.

NF Orchestration. NFs orchestration is the service-oriented interface for users to describe their demand through carefully designed API and realize cost-effective network function auto-scaling through a scheduler.
Network functions are divided into three planes:

Service Computing (SC): SC plane generates rules for instructing how the packet is processed based on the service logic of NFs, local NF-specific states, and customer preference. For a specific NF, multiple SC nodes constitute a reliability group (referred to as SCG) and synchronize the NF state to prevent state loss in case any SC node fails. It ensures the consistency and portability of the state in a SCG by redirecting the packet of rule requests to traverse all the SC nodes to synchronize states. We call this synchronization method *packet-pass-through*. Based on the operational experience, three nodes are sufficient for processing the state read request of a normal tenant.

Packet Processing (PP): The PP plane receives and processes packets based on rules generated by the SC plane. Each PP unit includes a parse-match-action table, which caches rules to instruct the PP on how to handle packets. The PP unit caches rules on-demand, ensuring quick readiness for scaling up and failover. When an incoming packet matches a rule, the PP node processes the packet according to the action. If no matching rule is found, the packet is forwarded to the SC plane. The rule consists of matching fields and actions with parameters. Matching fields specify which packet fields are used for matching and matching what value. If a flow matches with the value, it performs actions using parameters on packets of this flow. For example, in the case of Source NAT, the match fields of the rule are <Source IP, Source Port, Destination IP, and Destination Port>, and the action is modifying the source IP and port with substitutes. This design allows us to flexibly construct the processing pipeline by orchestrating PP units along the traffic path.

Fabric Master (FM): The FM plane is responsible for managing the interconnection between SC and PP instances, as well as facilitating communication among PP nodes. It avoids direct communication between any two nodes of the SC and PP planes. FM dispatches rule requests to SC nodes and returns the generated rules to PP nodes. When a new request arrives, it is initially forwarded to any arbitrary FM node. Subsequently, FM routes this request to the relevant SC nodes. After the decision-making process, which involves computing the rule, both the request and its corresponding rule are returned to the FM plane. Finally, they are delivered back to the originating PP node. Additionally, FM caches replicas of rules for incoming requests to alleviate the burden on the SC plane caused by repeated requests. For instance, if PP nodes crash, they will reboot and re-request the rules for active flows. Similarly, when a new PP node joins, some existing flows may be reassigned to this new node to distribute the workload evenly, triggering re-requests to SC. The rule storage in FM reduces the burden on SC by handling these repeated requests.

Based on the disaggregated architecture, NF developers only need to design and program the service logic, generating the processing ruleset using the API provided by CyberStar. This ruleset can then be installed into PP nodes. This approach simplifies the development process, as developers focus solely on the service logic, while CyberStar handles the deployment and execution of the ruleset across the lifecycle of the NF.

4 Elastic Scalability

4.1 Scalability of SC Plane

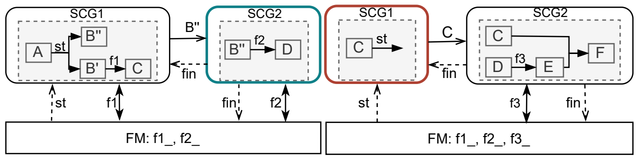
In the SC plane, we execute NF-specific service logic. Given the exponential growth in traffic and tenant demands, introducing multiple SCGs for service computation can significantly improve overall processing capacity and guarantee service quality. However, service computation needs NF-specific states locally for fast access, and the state consistency across all SCGs is crucial for correct rule generation. Any state modification triggered by an arriving request should be synchronized among multiple SCGs. With the increase of SCGs, the task of state synchronization can slow down service computation and limit the scalability of the SC plane. Therefore, we focus on how to realize effective state partition and state synchronization among SCGs.

State Partition. It is notable that the state can be partitioned based on the tenants because tenants are independent from each other in practice. This insight allows the partition of states into numerous SCGs for support cloud-scale traffic. By partitioning the state, consistency is maintained within each SCG using *packet-pass-through* as introduced in §3.2, and synchronization among SCGs is only required during scaling events. We further categorize NF state into two types: per-flow state and shared state. The per-flow state is only accessed by the packets of one flow. The shared state consists of structures or objects that are accessed or modified by multiple flows. Shared states with commutative properties[1] can be effectively partitioned. For instance, operations such as removing elements from a set can exhibit commutativity under specific conditions[2]. Taking Source NAT as an example, the available source address-port pool serves as a shared state, utilizing a set structure, with entries like <114.114.1.2, 1024 - 4096>. By partitioning the set into subsets like <114.114.1.2, 1024 - 2048> and <114.114.1.2, 2049 - 4096>, we can distribute them into two SCGs. Requests can also be dispatched across these two SCGs using hashing. This allows each SCG to independently allocate address-port pairs to incoming requests, ensuring efficient and independent operation. During the scaling-in event, the two sets can be merged back together.

State Synchronization during Scaling. State partitioning and merging can align with scaling-out and scaling-in operations, respectively. During a scaling-out event, the shared state can be partitioned and distributed across multiple instances.

[1]Commutativity refers to a property of operations where the order of applying the operations does not affect the final outcome. In other words, if two operations h and g are commutative, then applying h followed by g yields the same result as applying g followed by h. Mathematically, this can be expressed as $h(g(x)) = g(h(x))$ for any input x.

[2]Specifically, when the elements to be removed are distinct and present in the set, the removal operations are commutative.



(a) Scale out (b) Scale in

Figure 3: Example of scaling event in SCG. The capitals, e.g., A, B, C, represent shared state values. "fx" signifies a request set, while "fx_" indicates the corresponding returned rules.

After partitioning, the state can be updated independently. During a scaling-in event, these partitions can be merged and integrated into one state, even if each partition has been modified. The scaling event is a collaborative effort between SCG and FM, involving both scaling out and scaling in procedures. Scaling out is triggered when requests exceed the computation capacity of SC instances, and scaling in occurs when resources can be released. An example process is demonstrated in Figure 3. At the beginning of scaling out, FM sends a start signal (st) to SCG1. Upon receiving "st", SCG1 partitions state A into B$'$ and B$''$ based on partition strategy (e.g., bipartition for a set), moving state B$''$ to SCG2. At SCG1, the following arrived requests set (f1) can update its corresponding state B$'$. After receiving state B$''$, SCG2 sends "fin" to inform both SCG1 and FM that the state is ready. FM then dispatches new requests to SCG2, and SCG1 can delete state B$''$. In the case of scaling in, when it is determined that SCG1 needs to be released, the scaling in event is triggered. At the beginning of scaling in, FM sends a signal "st" to SCG1, instructing it to start to synchronize its states with SCG2. Meanwhile, FM forwards all following requests (f3) to SCG2. SCG1 receives "st" and synchronizes state C to SCG2. SCG2 receives state C, merges it with state E (D updated based on f3), and sends a "fin" signal to notify FM and SCG1 that synchronization is complete. Finally, SCG1 can be released.

During the procedures of both scaling out and scaling in, arrival requests can be processed continually with no need to wait for the complementation of state synchronization. Requests can be classified into two categories: requests arrived before scaling is triggered and newly arrived requests. For requests that arrive before scaling is triggered, we apply the rule cached in FM, as explained in §3.2, to process the requests. The cached rules minimize the impact of SC scaling. Specifically, for the scaling of PP plane, though the same requests might arrive repeatedly, FM can still respond based on the corresponding cached rules. For newly arrived requests during scaling our/in events, FM forwards the new requests to an SCG responsible for state partitioning/merging. All state modifications, including partitioning, merging, and updates caused by incoming requests, are completed within a single SCG. This approach ensures consistency by avoiding state write operation across different SCGs, eliminating the need to pause request processing.

4.2 Elastic PP Plane

In PP Plane, we provide high-speed packet processing shared by multiple NFs and tenants.

Modular Packet Processing. We introduce modular packet processing to unify the management of NFs. Since NFs follow the same pipeline match-action structure, the packet processing can be constructed as a pipeline built on a series of match-action units. Each match-action unit is defined as 1) Parse: extracting the key from the packet header based on protocols; 2) Match: looking up a flow table based on the key (flow-id or tenant-id) using wildcards; 3) Action: modifying/forwarding/dropping packet and/or updating local statistics. We deploy the match-action unit at one independent ECS instance. By doing so, we can flexibly construct the processing pipeline by organizing PP nodes. Instances are added along the traffic path to extend the pipeline depth. Meanwhile, each stage of the pipeline is separately scaled out by adding new instances. Due to states being maintained at the SC plane, PP instances only request rules on demand for arrival flows. This prevents traffic from halting to wait for large chunks of state migration during scaling events.

Rapid Scaling of PP Plane. The subsequent challenge involves enabling tenants to rapidly access new PP instances during scaling operations. Initially, we review how tenant traffic is directed to the PP plane. NFs and the applications of cloud tenants are deployed in different VPCs. Tenants access NFs based on the general VPC-to-VPC (Inter-VPC) communication mode. Given a task of accessing NFs, ECS creates and manages an Elastic Network Interface (ENI) [8, 21–23, 66], which represents the IP address and tenant id of the ECS instance used for network communications at the host level. For instance, if VPC *a* requires communication with VPC *b*, ECS within VPC *b* should create an ENI with an address (e.g., 192.168.10.3/24) and this address belongs to VPC *a* subnet (e.g., 192.168.10.0/24). Then, ECS within VPC *a* can send traffic to ECS within VPC *b* by taking the address 192.168.10.3 as the destination.

CyberStar leverages *ENI-bonding* to allow ECS instances of a tenant to access multiple PPs. ENI-bonding is a technology that enables ENIs attached to multiple ECS instances (in this context, PPs) to share a primary IP address. During establishing ENI-bonding, an ECMP group is generated that uses the primary IP as the destination and includes all ENI members in the ENI group as nexthops. The ECMP routing entries corresponding to these ENIs are loaded into the vSwitch where the tenants' ECS instances reside. This ensures tenant traffic can be directed to multiple PPs. By adding more ENIs connected to PPs into the ENI-bonding group, traffic can be redirected to new members, allowing the PP plane to scale out efficiently.

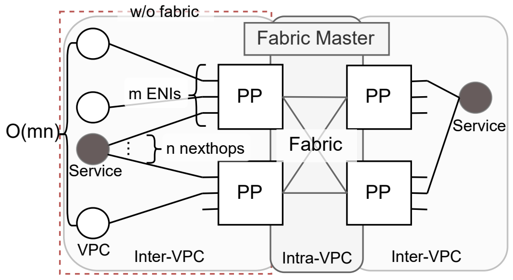


Figure 4: The fabric bridges the client VPC to Service by setting up the connection between two PP instances. Before introducing "Fabric", the number of clients who can access Service is constrained to $O(mn)$ as shown in the dotted box.

5 Fabric Master

The intricate interconnections between SCs and PPs, as well as cloud-based connections, affect scalability. We introduce the plane of Fabric Master to decouple the SC and PP planes.

5.1 Untangle Scaling of SC and PP

Problem. In the 1st Generation of CyberStar, PP instances request rules from different SC instances. For reliable requests, each PP node must establish multiple long-term, one-to-one connections with various SCGs. Consequently, each PP node needs several connections to dispatch requests to different SC instances. These connections tightly couple the SC plane and PP plane, resulting in inefficient scaling and failover of SCG. For scaling events, several connections should be established when new instances join and released when they leave. In the case of failover, the PP node needs to be aware of both primary and backup nodes within a SCG and perform a fast migration of connections from the primary to the backup node. Both the establishment of connections and migration processes cause high complexity of management and non-negligible wait time for the traffic involved.

Decouple SC and PP with Fabric Master. Fabric Master (FM) serves as the intermediary connecting the SC plane and PP plane which shields distributed SC structure for PP plane. Both SC instances and PP instances establish connections with FM. FM maintains connectivity and sessions so that members of the SC and PP planes can change and only requires to notifying FM. For SCGs scaling, PP instances remain unaware of any changes in the SC plane. During PP plane scaling, the process involves establishing or closing connections with FM without interrupting the SC plane.

5.2 Improve Tenants Accessing

Problem. CyberStar operates in cloud environments and distributes NF instances across high-density deployed ECSs[3]. In the 1st generation of CyberStar, tenants access cloud services

[3]A computation server can host a substantial number of containers for commonly used configurations, reaching up to O(10^3) [7, 50].

through NFs along the following path: tenants' VPC to PP instance to cloud service, as illustrated in the left dotted box of Figure 4. Each ECS instance can support only up to m ENIs, where m is no more than 100 [8, 21–23, 66], so the number of tenants who can simultaneously access cloud services is no more than 100. Even though a cloud service can connect to up to n PPs no more than 64 [61, 68] in our cloud, these one-hop connections only support $O(mn)$ tenants accessing a cloud service through PP plane[4].

However, this limitation cannot be resolved by merely increasing the number of ENIs supported by ECSs due to the requirements of searching tables. The virtual switch (vSwitch) running within the hypervisor has a limited buffer size, which restricts the size of tables, including route tables of different VPCs and mapping tables from ECS to physical machines. If the number of ECS hosted on each physical machine is i, and each ECS is attached to m ENIs, with n next hops to a destination, the routes maintained by vSwitch are $O(imn)$. The size of these route tables impacts route lookup efficiency, memory consumption, and the time required for migrating ECSs, which further limits the scale of tables. Therefore, increasing the number of ENIs is an impractical solution for improving tenant access to cloud services.

Fabric Abstraction for Interconnection of PP Instances. We introduce an abstraction of "Fabric" based on ECSs, which is an effective solution for improving the tenants accessing capability through CyberStar. As shown in Figure 4, Fabric is a virtual full mesh connection built on PP nodes. Based on Fabric, tenants can access cloud services by accessing any PP nodes. The connections of inter-PPs are not thus confined because *ECS instances belonging to the same VPC can connect without the requirement of extra ENI*.

Fabric is managed by FM. Fabric can be implemented by extending with a forwarding entry in the PP unit which records the next hop of a PP node. Since PP nodes should be stateless, the routes among PP nodes are maintained and updated by FM. If there is no route at the ingress PP when the packet of a flow arrives, the packet is sent to one of FM nodes to be forwarded to the next hop PP. Meanwhile, the corresponding route is installed into the PP node, indicating how to forward the following packets of this flow.

6 Deployment Flexibility

To deliver network function services effectively, a significant challenge is accommodating the diverse requirements of users, who often demand customized features.

6.1 Adoption of Heterogeneous Hardwares

To remain at the cutting edge, CyberStar allows managing heterogeneous hardware resources in the cloud to improve performance. We provide two approaches: in-depth integration for the general accelerator, e.g., DPU, and software/hardware co-design to accelerate packet processing.

[4]Cloud service connects to n PP instances, and each PP serves m tenants.

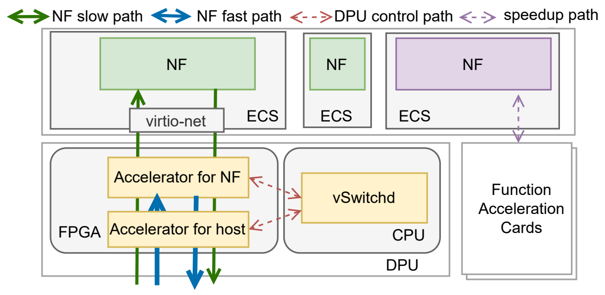


Figure 5: Architecture of hardware/software co-design. For DPU, the vSwitch and NF match-action (M/A) are offloaded. For function acceleration cards, the specified features are offloaded with the help of vSwitch and PP.

Network Function Acceleration. For the connection-oriented network functions, e.g., NAT and LB, we can maintain high scalability through PP nodes. However, the CPU-based packet processing has an intrinsic restriction on the throughput [40, 78]. In the virtualized environment, the vSwitch in the hypervisor performs packet sending and receiving among the ECSs and physical devices, occupying extra physical resources and leading to long latency and low throughput. The match-action unit has similar semantics with general-proposed hardware flow tables. Therefore, we seek to use widely deployed DPU/SmartNIC devices to offload the match-action unit, achieving $\sim 10\times$ performance gain.

The routing-oriented network functions, e.g., Layer-7 LB, are usually computation-sensitive. These network functions involve the complicated processing of encrypting and decrypting, key exchange, etc. As shown in Figure 5, to process large concurrent queries in a short time, we apply the acceleration cards, which can be deployed in remote machines.

Offloading vSwitch and NF Processing. The challenge is that DPU is applied for applications running over bare-metal servers instead of the virtualization domain of ECS [16, 17, 30]. To this end, we develop vDPU through the virtio device, a PCIe device following the virtio standard. It consists of two parts, the frontend in ECS and the backend in DPUs. We adopt virtio-block device for its compatibility with major operating systems and can be used by most VMs/containers without modification. We first offload the full vSwitch forwarding data plane and control plane into DPU. The offloading lib supports flow operation interfaces, such as rte_flow for DPDK and verbs for RDMA. And then, we offload the match-action processing into DPU.

Our DPU adopts *off-path* model [51, 76]. As shown in Figure 5, the software of DPU is a vSwitch management process (called vSwitchd) running by the embedded CPU on the DPU, which manages the hardware resources. The hardware part is an acceleration engine for the host used as the vSwitch forwarding data plane. The NF match-action is offloaded into the acceleration engine for ECS which is the fast path of ECS-based network function. When encountering elephant flows, SC node labels the corresponding rules in PP. The PP node calls the APIs, offloads the rule to vDPU, and waits for the response from vDPU. If successful, the DPU can directly process the remaining packets of the flow. It recycles the resources of network connections by periodically tearing down idle connections.

6.2 Live Migration

Leveraging heterogeneous hardware resources does not impact the flexibility of CyberStar under live migration technology. In this context, live migration refers to the process of moving a tenant's traffic from one NF instance to another without disrupting the traffic. We explain how CyberStar realizes the live migration between heterogeneous resources for computation and packet processing. The main idea is to decouple the address related to new hardware devices. To this end, we apply a virtual ENI as the interface for network functions to be implemented in the heterogeneous devices. CyberStar assigns the same virtual ENI for newly deployed devices and current NF instances. The controller configures and records the routes to the device in the vSwitch route table. Notably, the route table is customized by adding a "location" field. The client accesses the cloud server through NFs by using the virtual ENI. Once CyberStar identifies that the workload can be processed by the new hardware, it expands the bonding ENI to add the hardware-enhanced nodes and then removes the bonding ENI from the ECS nodes. As a result, during the process, the flows landed on the ECS nodes do not need to go through the SC plane, because the ECS nodes already know how to deal with the flows according to the stored forwarding actions. The flows landed on the hardware-enhanced nodes will go through the SC plane to get the action. Therefore, we can limit the flows that are sent to the SC plane and avoid the impact of burst traffic.

7 High Resource Utilization and Reliability

CyberStar adopts a hierarchical decision-making approach to achieve high resource utilization and maintain high reliability. With a global view, the NF orchestration realizes cost-effective network function auto-scaling by monitoring long-term resource utilizations, deciding when scaling events are triggered, and determining how tenants' traffic is dispatched into ECSs. In each NF instance, we deploy a rate management, that achieves local resource-sharing fairness among tenants and work conservation, effectively utilizing idle resources.

7.1 Global NF Orchestration

Network function auto-scaling is the key for CyberStar management platform to achieve high resource utilization and maintain high reliability. For achieving high resource utilization, the design of PP units has the potential to achieve effective resource sharing. Notice that there are three dimensions of loads: traffic, tenants and network functions. Besides the well-known dynamic arrival of tenants and their traffic, the resource consumption of different network functions varies. For example, VPN is computation-intensive, while NAT and LB are bandwidth and memory-intensive.

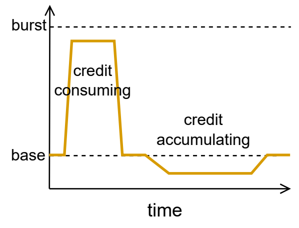
To maintain high reliability, we focus on minimizing the impact of "poison" requests from tenants. If a particular request happens to trigger a bug that causes the system to failover, then the caller triggers a cascading failure by repeatedly trying the same request against instance after instance until they have all fallen over. We adopt *Shuffle sharding* [67], an effective method for segregating tenants' workloads by distributing traffic across multiple instances with minimal overlap to isolate the fault domains.

The NF auto-scaling problem is usually defined as a *sequential decision problem* [12, 53]. Given cloud-scale workloads, the corresponding Integer Linear Programming (ILP) has tens of thousands of variables and constraints. CyberStar initially apply the heuristic bin packing algorithms [20, 32] to solve the problems. However, they fail to achieve long-term low resource utilization because the dynamic requests of NFs make the problem a *multi-stage decision problem* rather than a *one-shot decision problem*. Moreover, the heuristic algorithm needs to manually determine the water level of the maximum ECS resource usage from the (conservative) operators. To address the problem, we apply a data-driven Deep Reinforcement Learning (DRL) approach. By using DRL, we can leverage the historical information of traffic to optimize long-term utilization of ECS resources effectively and reduce the reallocation. We also solve the problems when deploying the algorithm in practice, like large action space and long validation time of sharding. The details are provided in Appendix A.

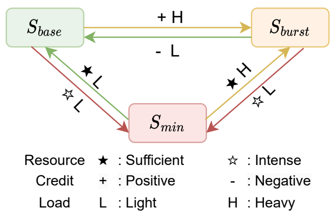
7.2 Local Rate Management

When flows belonging to multiple tenants are aggregated into a dedicated ECS, resource competition caused by transit flows is unavoidable and cannot be controlled by global orchestration. To reduce interference among tenants (ensuring high reliability) and avoid wasting idle resources (ensuring high resource utilization), we propose resource-aware rate management to achieve both (i) *Fairness sharing.* The excess resource consumption by a tenant under heavy load should not exceed its amount of abdicated resources under light load *in the long term*; (ii) *Work conservation.* Each tenant is permitted to utilize resources beyond its specification to process burst traffic when other tenants have lower processing demands.

The standard token-bucket (STB) mechanism is a simple traffic-shaping approach that permits bursts but strictly bounds them [46]. It ensures that a system or network does not exceed a specified fixed rate of traffic. In CyberStar, we leverage



(a) Credit variation



(b) State transformations

Figure 6: (a) The credit varieties with the state changing. The credit is consumed when S_{burst} while accumulated when S_{min}. Base and burst are thresholds, limiting the traffic arrival rates at S_{base} and S_{burst}, respectively. (b) State transformation determined by ECS available resource, the tenant credit, and tenant traffic load.

the STB mechanism to smooth the tenants' traffic entering an ECS. Furthermore, we dynamically transiently the rate allocated to each tenant so that this rate allocation can fully utilize available resources, enabling work conservation.

We define the traffic of the tenant can be one of three states: S_{min}, S_{base} and S_{burst}. These states represent different traffic rate control levels for tenants, corresponding to upper bounds of resource utilization. In other words, each tenant can utilize the resources according to their workloads as long as the traffic arrival rate does not exceed the designated threshold for a specific state. For equal-weighted fair sharing, the threshold of state S_{base} is set as the maximum processing rate divided by the number of tenants[5].

We utilize the credit to evaluate the resource utilization over time. A tenant's credit accumulates when it is under-provisioned and is consumed when it is over-provisioned, as illustrated in Figure 6a. Specifically, when the traffic arrival rate exceeds the processing capability of allocated resources (denoted as *base*), the tenant is over-using the resource, and the credit is continuously consumed until its workload decreases. Conversely, when the traffic arrival rate does not exceed the *base*, the tenant is under-using its resources, and the credit is continuously accumulated. The credit balance of a tenant can be negative optionally, allowing the tenant to go into debt to support heavy loads when there are adequate remaining resources. If a tenant's credit is depleted and there is an intense remaining resource, it will not be allowed to consume additional resources.

At the beginning of the allocation of an ECS to a tenant, the tenant is in the state S_{base}. The state transformation is shown in Figure 6b and illustrated as follows:

- In state S_{base}, if the tenant's credit is positive and its workload is heavy (i.e., arrival rate exceeds the rate threshold), the state switches to S_{burst}, allowing the tenant to use the resources yielded by other tenants. If the remaining resource is abundant and its workload is light, the state switches to S_{min}, so the tenant yields the idle resources.
- In state S_{min}, when the available resource is adequate, the state can switch to S_{base} when the tenant's traffic is light (i.e., the arrival rate is no more than the threshold), or to S_{burst} when the traffic is heavy.
- In state S_{burst}, when the tenant's traffic becomes light, the state can switch to S_{min} if the resource is intense, or to S_{base} if its credit is positive.

[5]Practically, the threshold can be set according to the bandwidth that the tenant subscribes to.

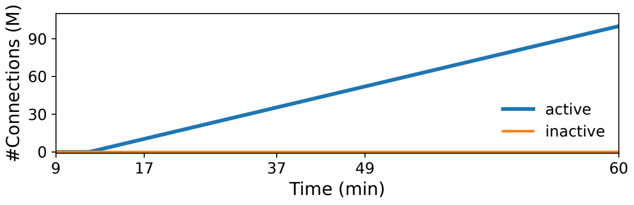


Figure 7: Demonstration of the scalability of CyberStar.

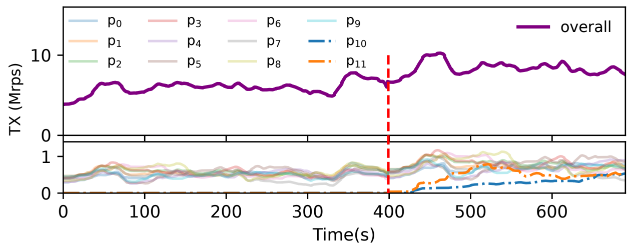


Figure 8: The transmission rate (TX) of PPs when a scaling event occurs.

8 Evaluation

CyberStar has been deployed and is publicly available in our cloud. In this section, we demonstrate the online performance of CyberStar and evaluate the proposed algorithms through a testbed with realistic tenant traffic demands.

8.1 Elasticity

We demonstrate the elasticity based on CyberStar's production deployment.

Scaling Ability. We evaluate the elasticity of CyberStar in the real system. Taking the load balancer (LB) as an example, it produces load balancing for the clients' traffic to access the cloud service. We show the ability of the supported connections to validate the scalability of CyberStar. We launch client and service clusters to test the connection establishment ability. Each cluster consists of 35 ECSs, and each ECS is equipped with 32 vCPUs, 128GB memory, and 15Gbps network bandwidth. The clients launch the connections with a speed of 35 thousand connections per second for 60 minutes. As shown in Figure 7, the overall active connections increase to 100 million. During this time, all connections are successively established and maintained actively.

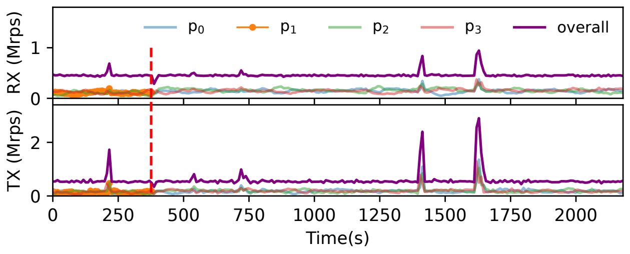


Figure 9: The arrival rate (RX) and transmission rate (TX) at a cluster with four PPs.

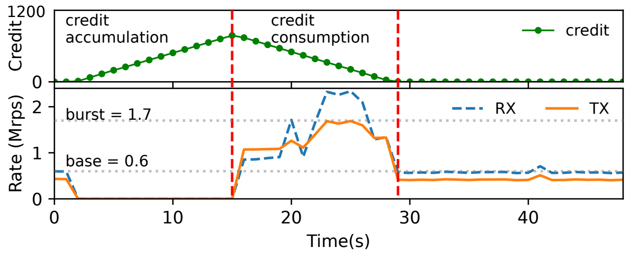


Figure 10: Rate limitation based on variation of credits.

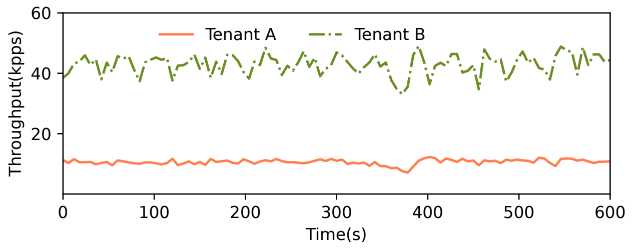
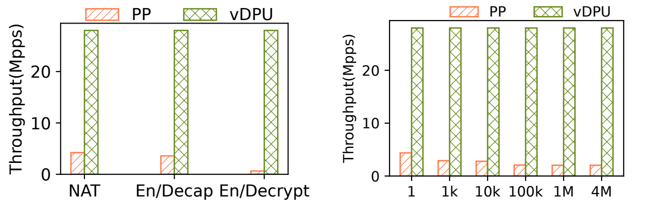


Figure 11: Traffic rate of tenants facing ECS failure.



(a) Throughput of different actions (b) Throughput with different number of flows

Figure 12: Performance of vDPU acceleration.

Scaling Efficiency. We collect statistics from ten scaling events to evaluate the scaling efficiency of CyberStar in the production environment. The time is counted when the scaling signal is received and stops when the traffic arrives at the new ECSs as shown in Figure 8. The scaling events are completed within a few seconds, and it is not correlated to the group size of new ECSs. Furthermore, we collect the throughput of a group that consists of 12 PP (marked as $p_0, \ldots, p_{11}$) to demonstrate the procedure of scaling. As shown in Figure 8, each node of this group reports its throughput to the monitor every 10 seconds. At first, this group has 10 PP nodes, and the scaling event is triggered at ∼400s. The scaling is completed within 30 seconds, and the PP p_{10} and p_{11} receive the requests. With the p_{10} and p_{11} joining in, the cluster is extended to 12 members and can process more traffic.

Burst Processing. To evaluate the capacity to process burst, we collect realistic traces of 36 minutes four PPs from a cluster. As shown in Figure 9, the requests arriving rate at the cluster is about 0.5 Mrps (requests per second). The maximal request arrival rate bursts up to 2× compared to the arrival rate. The packets in three observed bursts and other small bursts are absorbed efficiently.

To verify the efficiency of rate management, we analyze the credit variation with requests' arrival as shown in Figure 10. The base and burst requests process rates are 0.6 Mrps and 1.7Mrps, respectively. The credit is accumulated from 2s to 15s as the PP request arrival rate is lower than its base rate. The credit is consumed from 15s to 29s during the burst arrives. Meanwhile, the burst is suppressed under the burst rate and the state is in S_{burst} when the credit is sufficient. The credit is run out starting from 29s, and the tenant's state switches to S_{min}, allowing its sending rate not to exceed its base rate. Based on this, the credit can efficiently evaluate and control resource utilization by counting the arrival traffic.

Failure Recovery Efficiency. CyberStar can handle the ECS instance failure. When a PP node fails, the user traffic can be dispatched to the other healthy nodes. To assess the failure on tenants' traffic and validate the failure recovery efficiency, we collect workload traces of VNF requests in an available zone that encountered failover of one of their ECSs. As shown in Figure 11, the failover of one ECS happened at ∼390s, and the PPS of two tenants A and B slightly dropped and recovered in a short time. The result shows that CyberStar can seamlessly handle the redirected traffic from the failed ECS without causing disconnection of the workflow. We also evaluate the failure recovery in a cluster of four PPs. A PP p_1 crashed at ∼273s as shown in Figure 9. The overall received requests rate is reduced slightly for this PP crashing. After a few seconds, the traffic is dispatched to healthy nodes and the received and transmitted requests rate reverts.

8.2 Hardware Acceleration

The NFs deployed in ECSs offer throughput and latency that are comparable to those deployed directly on single bare-metal machines, thanks to the introduction of new trends in bare-metal cloud, which achieves native CPU and memory performance, along with para-virtualized I/O with minimal overhead [76]. In CyberStar, we leverage multiple ECS hosting different physical machines to split the processing burden and complement overall throughput equal to or even higher than single bare-metal machines achieved. The throughput of a single flow and the processing latency are determined by the computation power of a single entity. Therefore, we evaluate the single-core performance of ECS with and without vDPU acceleration to check whether the vDPU can complement processing latency and throughput. The baseline is the PP performance, where all packets go through the NF slow path.

Latency. We measure one-way latency between the traffic generator and PP by sending 1 million 64-byte packets sequentially over active TCP connections. When all packets go through vDPU, it achieves an average delay as low as 20.587μs, with a P99.9 delay $\sim$22.401μs and tail latency of 46.801μs. The latency diminishing is a benefit of the shorter processing path and high-speed hardware.

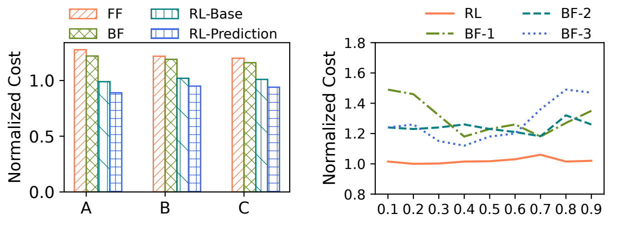
Throughput. We evaluate the performance gain through DPU offloading for network functions. In the experiment, we perform the actions of NAT, encapsulation/decapsulation, and encrypt/decrypt as they are basic operations of deployed NFs. First, we offload a rule into vDPU to evaluate the processing efficiency for different actions. As shown in Figure 12a, the throughput of vDPU performing NAT and encapsulation/ decapsulation actions is 6.6$\times$ and 7.8$\times$ in comparison with ECS, respectively. For the compute-intensive actions encrypt/decrypt, its throughput improves by 43.48$\times$. Then we test the performance with the increasing number of flows. As shown in Figure 12b, the throughput decreases with the increasing flow table size. The flow table size affects the performance of CPU-based PP for the lookup operation while always maintaining stable performance for vDPU.

8.3 Resource Utilization

In the experiment, we use realistic traffic demands, reliability strategies, and price models from production networks. We collect three data sets with different scales, i.e., A, B and C, listed in the ascending order of the number of tenants. A and B contain traffic records from hundreds of tenants while C contains traffic records from over 1000 tenants. The number of available ECSs is set to 100 for each test, and the ECS utilization threshold is set to 50%. Each data set contains traffic records from the realistic cloud for several weeks.

We compare our algorithm with the enhanced version of the First-Fit algorithm (FF), which is an online algorithm for the multi-dimensional vector bin packing problem [32] [20], and a specific weighted Best-Fit (BF) algorithm initially used in the production network. As shown in Figure 13a, compared with the FF and BF algorithm, the DRL-Base algorithm can achieve $\sim$15%-25% lower cost. This result verifies that the DRL algorithm can effectively utilize historical information to learn delayed rewards. The DRL agent can automatically explore the search space without the need to manually design and tune heuristics with human experts. Compared with the DRL algorithm without traffic prediction, the DRL agent combined with traffic prediction can achieve $\sim$5%-10% lower cost. This result shows that traffic prediction can effectively help the DRL agent to make better decisions.

As depicted in Figure 13b, we conducted a performance evaluation of both the DRL algorithm and BF algorithms using varying combinations of mixed NF requests. The datasets employed in this experiment comprised requests for two different NFs, namely NAT and SLB. We systematically adjusted the percentage of NAT requests from 10% to 90% and randomly modified the weights of the BF algorithms ten times (Figure 13b only displays three out of ten iterations with better results). None of them managed to outperform the DRL approach. This further underscores the challenge of tuning BF weights to consistently yield superior and stable results, especially when dealing with diverse types of NF requests.



(a) DRL performance under different traffic distribution
(b) DRL performance under mixed NF loads

Figure 13: DRL outperforms FF/BF under different traffic loads and traffic prediction can improve DRL performance.

9 Experiences and Lessons

Challenges in Meeting Customized Demands. Before the integration of multiple NF services into CyberStar, each type of NF service operated independently under separate departments. They provide their own NF service to users but those NF services lack much collaboration or information sharing among others. While each NF service may have been optimized within its domain, the combined NF service chain presented to users is not the most efficient or effective. From a user's perspective, selecting the proper NF services from a diverse range of options to meet their specific application demands can be challenging without expert guidance.

A real-world example highlights these challenges. Enterprise customers often choose NF services from different *regions* for the lowest prices, especially when they lack expertise in specific areas. A customer establishes a service chain of firewall-NAT-load balancer, but it chooses a firewall and a NAT at Region R1, and a load balancer at Region R2. This could result in user traffic taking roundabout routes through different regions (R1-R2-R1), leading to significant delays. However, since no service in the chain is aware of the complete forwarding path, they cannot take any action to eliminate these roundabout routes. When users complained about poor performance, each service would independently troubleshoot its own issues but find nothing wrong. Eventually, the root cause of the traffic roundabout was manually discovered through traffic tracing, and the ultimate solution is guiding users to purchase the services in appropriate regions with the help of experts.

CyberStar provides a global view for customized demands. Integrating NF services into a unified management platform provides service providers with a comprehensive view of service implementation and deployment, eliminating the barriers that existed before. In the case mentioned earlier, CyberStar establishes a closed-loop monitoring system, enabling end-to-end performance optimization and ensuring the best service quality. From a long-term perspective, with a wide range of NF services integrated into CyberStar, it becomes possible to intelligently generate specific plans for each user according to their customized demands or provide the global view and concise interfaces for users to select their preferred services.

Why Not Deploy on Kubernetes? The distinctions between Kubernetes (k8s) and CyberStar can be delineated based on several key aspects. First, k8s cannot deliver multi-tenant services, as each pod in k8s serves only one tenant, whereas CyberStar is explicitly designed to accommodate multiple tenants simultaneously. Secondly, for services with state persistence and synchronization requirements, k8s uses databases to store and synchronize states. However, using a database for state storage and synchronization may not meet the throughput and latency demands of business logic processing. Network function states need to be stored in memory. CyberStar leverages the packet-pass-through method to achieve fast state synchronization.

ECS Selection Perference. We structure the NF platform into three tiers to leverage cloud elastic resources, enabling rapid scaling with a few types of ECSs. However, efficiency and reliability necessitate specific preferences in ECS selection. We deploy loosely coupled network function components on ECSs with varying configurations to match specific demands for computational, memory, and networking resources. Given the distinct software suites used in SC, FM, and PP, we deploy SC and FM instances on compute-optimized ECSs, while PP instances are deployed on network-enhanced ECSs. Compute-optimized instances are optimized for applications requiring high-performance processors, making them suitable for SC, which handles service logic computing, and FM, which manages the connections among SC and PP instances. Network-enhanced general-purpose instances significantly improve network throughput and packet forwarding rates, making them ideal for the PP plane, which requires high-speed processing and forwarding.

10 Related Work

Network Function Virtualization. Network function virtualization has been widely studied in the last decade [42–44, 71], e.g., LB [56], VXLAN gateway [61], IPSec VPN [69]. Some studies on NF virtualization mainly focus on the design of specified functions [25, 57, 72]. However, addressing the diverse and extensive demands of network services often necessitates the creation of isolated clusters for each type of network function. This approach is cost-ineffective due to the lack of resource aggregation. Some efforts [34, 35, 44, 58, 60, 70] aimed to establish a more generalized NFV framework. They either categorize NFs as monolithic instances [58, 59] or additionally decouples NFs into two-tier architectures [15, 44, 64, 70]. Monolithic VNFs suffer from shortcomings like redundant development and optimization efforts on common tasks across different NFs. Works [35, 44, 64, 70] introduced state management mechanisms to improve scaling performance. Other two-tier works [15, 19] depart common processing from NFs to ease the management of separate NFs. CyberStar mainly focuses on factors that limit the scalability of cloud-native NF management including state, connection complexity and tenant accessing cloud service through NFs.

Performance Optimization. A large number of works focus on improving the performance of VNFs through different optimization points. One is packet delivery acceleration like Intel DPDK [39], ClickOS [55], NetVM [38, 75] that optimize the packet delivery from NICs to VMs and between VMs. Another line is hardware acceleration, e.g., FPGA, GPU, and P4 switch [28, 48, 63, 73]. Generally, hardware-based accelerators focus on NF characteristics to boost the performance [18, 42, 43, 56, 61, 62]. APUNet [36] and G-Net [74] make use of batching processing of GPU to enhance the throughput for NFs. ClickNP [48] designs the FPGA-based modular data plane to implement NFs quickly. FlowBlaze [63] uses the FPGA to accelerate the stateful NF data plane. The design of the match-action unit can be accelerated by the hardware flow table. CyberStar offloads the match-action units in the shared hardware accelerator, optimizing NFs and minimizing the involvement of the host CPU.

11 Conclusion

With the trend of migrating applications to the cloud, the traditional NF architecture based on the bare-metal server cluster is challenging to meet the elasticity requirements and low costs. We introduce CyberStar, a cloud-native network functions management platform to achieve high elasticity. We describe the architecture of CyberStar, which leverages the loosely coupled pooling resource to enable on-demand allocation for packet processing, service computation, as well as interconnection management through fabric abstraction. CyberStar has been deployed for over four years and is publicly available in our cloud.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and constructive feedback. This research is supported by the National Key R&D Program of China under Grant number 2022YFB2901502, National Natural Science Foundation of China under Grant No. 62172204, 62325205, 62072228, and Key R&D Program of Zhejiang Province 2023R5202.

References

[1] Satyajeet Singh Ahuja, Vinayak Dangui, Kirtesh Patil, Manikandan Somasundaram, Varun Gupta, Mario A. Sánchez, Guanqing Yan, Max Noormohammadpour, Alaleh Razmjoo, Grace Smith, Hao Zhong, Abhinav Triguna, Soshant Bali, Yuxiang Xiang, Yilun Chen, Prabhakaran Ganesan, Mikel Jimenez Fernandez, Petr Lapukhov, Guyue Liu, and Ying Zhang. Network entitlement: contract-based network sharing with agility and SLO guarantees. In *Proceedings of the ACM SIGCOMM Conference*, 2022.

[2] Alibaba Cloud. Elastic compute service. Elastic Compute Service. https://www.alibabacloud.com/en/product/ecs?_p_lc=1, 2024.

[3] Alibaba Cloud. How does cloud empower double 11 shopping festival. How Does Cloud Empower Double 11 Shopping Festival. https://resource.alibabacloud.com/event/detail?id=1281, 2020.

[4] David A.Maltz. Scaling challenges in cloud networking. In *Microsoft Research Summit*, 2021.

[5] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, et al. Providing SLOs for resource-harvesting VMs in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[6] AWS. Amazon. Amazon Elastic Container Service. https://aws.amazon.com/cn/ecs/, 2021.

[7] AWS. Amazon. What is Cloud Native? - Cloud Native Explained. https://aws.amazon.com/what-is/cloud-native, 2021.

[8] Azure. How to create a linux virtual machine in azure with multiple network interface cards. https://learn.microsoft.com/en-us/azure/virtual-machines/linux/multiple-nics, 2022.

[9] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM Conference*, 2011.

[10] 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)*, 2023.

[11] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. Online job scheduling in distributed machine learning clusters. In *INFOCOM Conference on Computer Communications*. IEEE, 2018.

[12] Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Otto Carlos Muniz Bandeira Duarte. Orchestrating virtualized network functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739, 2016.

[13] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014.

[14] Timm Böttger, Ghida Ibrahim, and Ben Vallis. How the internet reacted to covid-19: A perspective from facebook's edge network. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2020.

[15] Anat Bremler-Barr, Yotam Harchol, and David Hay. Openbox: A software-defined framework for developing, deploying, and managing network functions. In *Proceedings of the ACM SIGCOMM Conference*, 2016.

[16] Brad Burres, Dan Daly, Mark Debbage, Eliel Louzoun, Christine Severns-Williams, Naru Sundar, Nadav Turbovich, Barry Wolford, and Yadong Li. Intel's hyperscale-ready infrastructure processing unit (ipu). In *IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021.

[17] Idan Burstein. Nvidia data center processing unit (dpu) architecture. In *IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021.

[18] Jian Chen, Xiaoyu Zhang, Tao Wang, Ying Zhang, Tao Chen, Jiajun Chen, Mingxu Xie, and Qiang Liu. Fidas: fortifying the cloud via comprehensive FPGA-based offloading for intrusion detection: industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022.

[19] Shihabur Rahman Chowdhury, Haibo Bian, Tim Bai, Raouf Boutaba, et al. A disaggregated packet processing architecture for network function virtualization. *IEEE Journal on Selected Areas in Communications*, 38(6):1075–1088, 2020.

[20] Henrik I Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Multidimensional bin packing and other related problems: A survey. *Computer Science Review*, 2016.

[21] Alibaba Cloud. Instance family. https://www.alibabacloud.com/help/en/doc-detail/25378.htm?spm=a2c63.p38356.0.0.9c8bbe1avPJfMq#concept-sx4-lxv-tdb, Sep. 2017.

[22] Google Cloud. Creating instances with multiple network interfaces. https://cloud.google.com/vpc/docs/create-use-multiple-interfaces, Sep. 2022.

[23] Tencent Cloud. Use limits overview. https://intl.cloud.tencent.com/document/product/213/15379, Aug. 2022.

[24] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, 2017.

[25] 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)*, 2018.

[26] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[27] 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 *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[28] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. NICA: An infrastructure for inline acceleration of network applications. In *USENIX Annual Technical Conference (ATC)*, 2019.

[29] FD.io. VPP. https://wiki.fd.io/view/VPP.

[30] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[31] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-harvesting vms in cloud platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[32] Michael R Garey, Ronald L Graham, David S Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[33] Aaron Gember, Robert Grandl, Junaid Khalid, and Aditya Akella. Design and implementation of a framework for software-defined middlebox networking. *ACM SIGCOMM Computer Communication Review*, 43(4):467–468, 2013.

[34] Aaron Gember, Anand Krishnamurthy, Saul St John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Aditya Akella, and Vyas Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. Technical report, Technical Report, 2013.

[35] 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.

[36] Younghwan Go, Muhammad Asim Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. Apunet: Revitalizing GPU as packet processing accelerator. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[37] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, 2019.

[38] Jinho Hwang, K. K Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, 12(1):34–47, 2015.

[39] Intel. Data plane development kit, 2014.

[40] Intel. Receive-side scaling (rss). http://www.intel.com/content/dam/support/us/en/documents/network/sb/318483001us2.pdf, 2016.

[41] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[42] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[43] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, 2017.

[44] 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)*, 2017.

[45] Georgios P Katsikas, Tom Barbette, Dejan Kostic, Rebecca Steinert, and Gerald Q Maguire Jr. Metron:nfv service chains at the true speed of the underlying hardware. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[46] Jayakrishna Kidambi, Dipak Ghosal, and Biswanath Mukherjee. Dynamic token bucket (dtb): a fair bandwidth allocation algorithm for high-speed networks. *Journal of High Speed Networks*, 9(2):67–87, 2000.

[47] Neeraj Kulkarni, Feng Qi, and Christina Delimitrou. Pliant: Leveraging approximation to improve datacenter resource efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[48] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the ACM SIGCOMM Conference*, 2016.

[49] Defang Li, Peilin Hong, Kaiping Xue, et al. Virtual network function placement considering resource optimization and sfc requests in cloud datacenter. *IEEE Transactions on Parallel and Distributed Systems*, 29(7):1664–1677, 2018.

[50] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *ATC'22*, pages 53–68. USENIX Association, 2022.

[51] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM SIGCOMM Conference*, 2019.

[52] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[53] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015.

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

[55] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[56] 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 ACM SIGCOMM Conference*, 2017.

[57] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262, 2015.

[58] Open Source MANO. `https://osm.etsi.org`, 2023.

[59] OPNFV Project. `https://www.opnfv.org`, 2020.

[60] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A framework for nfv applications. In *Proceedings of the 24th Symposium on Operating Systems Principles (SOSP)*, 2015.

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

[62] Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafrir. Autonomous nic offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[63] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, et al. Flowblaze: Stateful packet processing in hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[64] 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)*, 2013.

[65] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the ACM SIGCOMM Conference*, 2017.

[66] Amazon Web Services. Elastic network interfaces. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html#AvailableIpPerENI, Mar. 2017.

[67] Amazon Web Services. Shuffle sharding: Massive and magical fault isolation. http://www.awsarchitectureblog.com/2014/04/shuffle-sharding.htm, Sep. 2017.

[68] Hua Shao, Xiaoliang Wang, Yuanwei Lu, Yanbo Yu, Shengli Zheng, and Youjian Zhao. Accessing cloud with disaggregated software-defined router. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[69] Jeongseok Son, Yongqiang Xiong, Kun Tan, Paul Wang, Ze Gan, and Sue Moon. Protego: Cloud-scale multitenant ipsec gateway. ATC. USENIX Association, 2017.

[70] 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)*, 2018.

[71] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the ACM SIGCOMM Conference*, 2017.

[72] Bo Yi, Xingwei Wang, Keqin Li, Min Huang, et al. A comprehensive survey of network function virtualization. *Computer Networks*, 133:212–262, 2018.

[73] Yifan Yuan, Yipeng Wang, Ren Wang, and Jian Huang. HALO: Accelerating flow classification for scalable packet processing in NFV. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019.

[74] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. G-net: Effective GPU sharing in NFV systems. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[75] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, KK Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proceedings of the workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 26–31, 2016.

[76] Xiantao Zhang, Xiao Zheng, Zhi Wang, Hang Yang, Yibin Shen, and Xin Long. High-density multi-tenant bare-metal cloud. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[77] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G Edward Suh, and Christina Delimitrou. Sinan: Ml-based and qos-aware resource management for cloud microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[78] Peng Zheng, Arvind Narayanan, and Zhi-Li Zhang. A closer look at nfv execution models. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking*, 2019.

A Details of NF orchestration

A.1 Problem Formulation

We define the problem as the VNF placement problem targeting minimizing the overall ECS cost. The notations are listed in Table 2 and the formulation is described as follows.

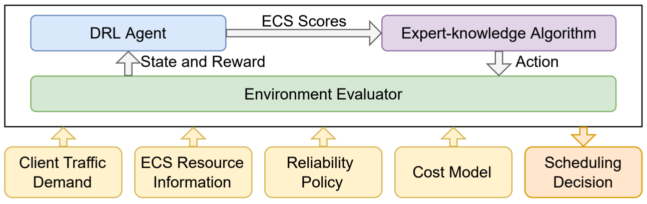


Figure 14: Workflow of NFs scheduler

Symbol	Description
V	Set of VNF types
D	Set of VNF requests from tenants
E	Set of ECSs
U	Threshold in shuffle sharding
O	Maximum number of shared ECSs allowed by two tenants
G^i	Number of ECSs to which the tenant requests i to be deployed
k_e	Cost of purchasing ECS e
x_e^i	Binary variable indicates whether request i is placed on ECS e.
y_e	Binary variable indicates whether ECS e is purchased
b^i, c^i, m^i	Bandwidth, computing and memory resource required by tenant request i
UB_e, UC_e, UM_e	Bandwidth, computing and memory utilization of ECS e
B_e, C_e, M_e	Bandwidth, computing and memory resource capacity of ECS e

Table 2: Notations for problem formulation

Objective. The objective is to minimize the total cost of used ECSs, which is determined by the price of all ECSs used to handle VNF requests of tenants, shown as Eq. (1). Each ECS is bought as a package of resources (CPU, memory, and bandwidth), and its specification determines the ECS price.

$$\min_{y_e} \sum_{e \in E} k_e y_e \tag{1}$$

$$\sum_{i \in D} \frac{b^i}{G^i} x_e^i \leq UB_e y_e, \quad \forall e \in E \tag{2}$$

$$\sum_{i \in D} \frac{c^i}{G^i} x_e^i \leq UC_e y_e, \quad \forall e \in E \tag{3}$$

$$\sum_{i \in D} \frac{m^i}{G^i} x_e^i \leq UM_e y_e, \quad \forall e \in E \tag{4}$$

$$G_{min} \leq G^i \leq G_{max}, \quad \forall i \in D \tag{5}$$

$$\sum_{e \in E} x_e^i x_e^j \leq O, \quad \forall i, j \in D, i \neq j \tag{6}$$

The constraints in Eq. (2)-(6) are explained below.

- Resource capacity constraints (Eq. (2), (3), (4)): The percentage of ECS physical resources reserved by all requests on it should not exceed the predetermined threshold U for each type of resources.
- ECS allocation constraint: (Eq. (5)): G^i indicates the number of ECSs used by tenant i, which can be formally defined as $\sum_e x_e^i = G^i, i \in D$. This value ranges from G_{min} to G_{max} to control the incidence caused by user "poison" requests at shuffle-sharding algorithm [67].
- ECS overlap constraint (Eq. (6)): The number of shared ECSs between two tenants should not exceed O. The value of O is determined by operators.

Note that Eq.(2), (3) and (4) are used to capture multi-dimensional resource attributes, while Eq.(5) and (6) are used to capture reliability attributes.

A.2 DRL Approach

The VNF placement problem can be defined as a sequential decision problem, where the agent needs to select a set of ECSs for tenant requests sequentially. For each tenant, the agent takes action, i.e., allocating ECSs for the tenant request, so as to maximize the long-term reward that can be inferred from Eq. (1). Note that the selection for the i-th tenant depends on the allocation for (i-1) previous tenants due to the constraints Eq. (2)-(6). Such sequential decision problems are often formulated using Markov Decision Process (MDP).

Markov decision process is a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function, $\mathcal{R}$ is the reward function, and $\gamma \in [0,1]$ is the discount factor. After taking action A_t on state S_t, the agent will observe the new state S_{t+1} according to the transition function, and the corresponding reward $\mathcal{R}_t$ according to the reward function. The objective is to find a policy $\pi(s,a)$ of selecting an action given a state so that we can obtain the long-term reward, that is the total sum of discounted rewards going forward $\mathcal{G}_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. Alternatively, we can learn estimates for the optimal value of each action $Q_\pi(s,a) = E_\pi[\mathcal{G}_t | S_t = s, A_t = a]$ (the action-value function), which is the expected future reward if taking action a at state s and following the optimal policy. The optimal policy can be easily found by taking the highest value of action-value function $Q(s,a)$ at state s.

For the VNF placement problem, the next state and reward can be obtained using an environment emulator as shown in Figure 14. The concrete definition of states, actions, and rewards for our VNF placement problem is given as follows.

- State: The i-th state encapsulates available ECSs and the resources required by the i-th tenant. Formally, $S_i =< \mathcal{B}, \mathcal{C}, \mathcal{M}, b^i, c^i, m^i > \in R^{3 \times |E| + 3}$, where $\mathcal{B} \in R^{|E|}$, $\mathcal{C} \in R^{|E|}$, $\mathcal{M} \in R^{|E|}$ are vectors indicating the bandwidth, computing and memory resources currently available on each ECS.
- Action: At the i-th state, the action is to select a set of ECSs to assign to the i-th tenant. As a result, the action space contains all possible combinations (of size from G_{min} to G_{max}) of available ECS at the i-th state.
- Reward: We design the reward so that it will guide our training algorithm to find an optimal policy for selecting an action given a state. In other words, if following the optimal policy for all the tenants, we can get the optimal long-term reward, which has minimum cost on ECS. Specifically, the reward for each action is set: 1) if the action contains G_{new}

Algorithm 1 Training Process

```
Require: $\mathcal{S}$: State; $N$: Number of epochs; $\varepsilon$: The parameter of $\varepsilon-$greedy
    strategy.
Ensure: The trained Q-network $Q(s,a,\theta)$
    Main routine
 1: Initialize $\theta, \theta'$ parameters for the Q-network and the target network
    respectively
 2: Initialize replay buffer, and the tenant set $\mathcal{T}$ to be empty.
 3: for $episode = 1,2,3,...N$ do
 4:     $\mathcal{T} = Updatetenant()$
 5:     for Each tenant Request $t \in \mathcal{T}$ do
 6:         $S_t = GetState(t)$
 7:         Generate a random number $\beta \in [0,1]$.
 8:         if $\beta < \varepsilon$ then
 9:             Select available ECSs with probability $\varepsilon$.
10:         else
11:             $A_t = FilteringAlgorithm(Q(S_t,a,\theta)\forall a)$, $A_t$ is a set of ECS
12:         end if
13:         Execute action $A_t$ in emulator and observe reward $R_t$ and new
            state $S_{t+1}$.
14:         Store experiences $(S_t, a_t, R_t, S_{t+1})$ for each ECS $a_t \in A_t$ into the
            replay buffer.
15:         Select a minibatch of experiences from the replay buffer
16:         Update the parameters for the Q-network using gradient decent
            based on the minibatch to minimize the loss in Eq. (7).
17:         After every $C$ steps, save the Q-network as the target network.
18:     end for
19: end for
    Subroutines
```

- Updatetenant(): Update the tenant set containing the tenants needed to be scheduled currently.
- GetState(i): Get the state $\mathcal{S}$ from environment about ECSs and request i.
- RandomAction(): Randomly explore the action space until finding an action that satisfies all the constraints.
- FilteringAlgorithm is shown in Algorithm 2

Algorithm 2 Filtering Algorithm

```
Require: $\mathbf{Z}$: The score of each ECS; $G_{min}, G_{max}$: The minimum and max-
    mum number of ECSs allowed to shuffle sharding; $k$: The parameter
    trades off complexity and feasibility.
Ensure: $A$: The set of ECSs.
 1: Resort ECSs in descent score;
 2: Select top $k$ actions in $\mathbf{Z}$ as the candidate ECSs set $\mathbf{Z}'$;
 3: Obtain all the combination ECSs $O$ in $\mathbf{Z}'$ with length in $[G_{min}, G_{max}]$;
 4: Remove the combination ECSs $A' \in O$ violate the constraints;
 5: Calculate the reward of each $A' \in O$ denoted as $R_{A'}$;
 6: $A = \arg\max_{A' \in O} \mathcal{R}_{A'}$;
```

new ECS e, then $\mathcal{R} = -\sum_{e \in G_{new}} k_e$, where the negative value means higher cost and lower reward. 2) if the action distributes the request to already used ECSs, the reward will be computed based on the water level of used ECSs as $\mathcal{R} = \sum_{e \in G_{new}} (UB_e + UC_e + UM_e)^{-1}$. The reward is normalized so that it falls in the range of [-1, 1].

Reinforcement learning (RL) learns the optimal policy $\pi(S,a)$ through episodes of interactions with the environment, where an episode is a sequence of state and action $(S_1, A_1, S_2, A_2, ..., S_T, A_T)$ and T is the number of current tenant requests. RL is particularly appealing for sequential decision problems in dynamic environments such as the VNF placement problem. Unfortunately, one challenge as mentioned above is the combination complexity of the action space. Furthermore, it is difficult to design a reward that could lead to a solution satisfying all the constraints in Eq. (2)-(6). To handle these issues, we simplify an action for our VNF allocation MDP corresponds to selecting only one ECS instead of a set of ECSs. Since each action corresponds to an ECS, the values for all actions Q(s, a) given a state s form the ranking of ECSs for the tenant request of the state s. The combinations of ECSs within top-k highest values are filtered further by a Filtering Algorithm to select a set of suitable ECSs satisfying constraints in Eq. (2) - (6). By doing so, we effectively reduce the action space of our MDP, and the Filtering Algorithm only needs to work with a constant number (k) of most potential ECSs produced by $Q(s,a)$.

In this paper, we modify the Double Deep Q-Network (DDQN) algorithm to integrate our Filtering Algorithm. The main idea of DDQN lies in three folds: 1) it represents $Q(s,a)$ as a deep neural network parameterized by θ and denoted by $Q(s,a,\theta)$. Here we use Multilayer Perceptron (MLP) to represent our Q-network $Q(s,a,\theta)$; 2) it uses a replay buffer $\mathcal{U}(D)$ of (S_t, A_t, R_t, S_{t+1}) to save experiences and improve the data efficiency; and 3) $Q(s,a,\theta)$ is saved periodically to a target network $Q(s,a,\theta')$. The parameter θ is learned by optimizing the following loss:

$$\mathcal{L}(\theta) = E_{(S_t,A_t,R_t,S_{t+1})\ \mathcal{U}(D)}[(R_t + \gamma * Z_t - Q(S_t, A_t, \theta)]^2 \quad (7)$$

where $Z_t = Q(S_{t+1}, A_{t+1}, \theta')$ is the target value calculated by the target Q-network (θ') for the state S_{t+1} and the action A_{t+1}. Here, the action A_{t+1} is drawn from the set of suitable actions (ECS) selected by the Filtering Algorithm. This is different from the standard DDQN where A_{t+1} is drawn from the Q-network $Q(s,a,\theta)$ instead of the Filtering Algorithm.

Offline Training. The parameter θ is updated using gradient descent by drawing sample tuples from the memory reply $\mathcal{U}(D)$. Our Training algorithm and the Filtering Algorithm are displayed in Appendix A Algo.1, and 2. The workflow of the scheduling system in Figure 14 summarizes the main components and the interactions between them in our RL approach.

Online Running. The online running of Algo.1 is triggered by three events, including new tenant arrival, ECS scaling down, and scaling up. The DRL agent accepts VNF requests from new tenants in real time and calculates the allocation result. The scaling down and scaling up events are detected by regularly checking the utilization of ECSs. Once the utilization of some ECSs is larger than the upper bound threshold, we need to scale up and reallocate resources to compensate for extra traffic. On the other hand, scaling down is needed when the utilization falls under the lower-bound threshold, indicating the low utilization of some ECSs. As a result, VNF instances will be reallocated and unnecessary ECS can be freed. The DRL agent periodically obtains state information of all ECSs from the platform. It stores each state transition, corresponding action and reward to periodically update the DQN model.