Triton: A Flexible Hardware Offloading Architecture for Accelerating Apsara vSwitch in Alibaba Cloud

Xing Li[†§★], Xiaochong Jiang[†★], Ye Yang[§★], Lilong Chen[†], Yi Wang[§], Chao Wang[§], Chao Xu[§], Yilong Lv[§], Bowen Yang[§], Taotao Wu[§], Haifeng Gao[§], Zikang Chen[§], Yisong Qiao[§], Hongwei Ding[§], Yijian Dong[§], Hang Yang[§], Jianming Song[§], Jianyuan Lu[§], Pengyu Zhang[§], Chengkun Wei[†□], Zihui Zhang[†], Wenzhi Chen[†□], Qinming He[†], Shunmin Zhu[‡§□]

[†]Zhejiang University [§]Alibaba Cloud [‡]Tsinghua University

ABSTRACT

Apsara vSwitch (AVS) is a per-host deployed forwarding component for instance network connectivity in the Alibaba Cloud. To meet the growing performance demands, we accelerated AVS by adopting the most widely used "Sep-path" offloading architecture, which introduces a separate hardware data path to speed up popular traffic. However, the deployment results prove that it is difficult to bridge the gap in performance and programming flexibility of the software and hardware data paths, resulting in unpredictable performance and low iteration velocity.

This paper introduces *Triton*, a flexible hardware offloading architecture for accelerating AVS. *Triton* stands out with a unified data path, where each packet passes serially through software and hardware processing to ensure predictable performance. For flexibility, *Triton* implements an elegant workload distribution model, which offloads generic tasks to hardware but maintains dynamic logic in software. Additionally, *Triton* integrates a series of cutting-edge software-hardware co-designs, including vector packet processing and header-payload slicing, to mitigate software bottlenecks and enhance forwarding efficiency. The deployment results compared to our prior solution based on "Sep-path" reveal that *Triton* achieves predictable high bandwidth and packet rate, and notably improves the connection establishment rate by 72%, with only 2μs increase in latency. More importantly, the flexibility and iteration velocity of the software in *Triton* will save development costs and bolster maintenance efficiency for cloud vendors.

KEYWORDS

Virtual switch, Hardware offloading, SmartNIC

1 INTRODUCTION

Apsara vSwitch (AVS) is a critical, per-host forwarding component in Alibaba Cloud, providing connectivity and management for computing instances like VMs, containers, and bare metals, akin to the open-source and other industry vSwitches [1, 36, 46, 47, 53]. Apart from the basic functions, AVS also supports advanced features for tenants, such as Traffic Mirroring [11, 19] and Flowlog [8, 16]. To achieve that, both the basic and advanced functions are facilitated by numerous predefined policy tables, necessitating efficient packet matching for action determination. Consequently, accelerating "match-action" and enhancing AVS forwarding performance remain our constant pursuit.

Over the past decade, we have integrated various optimization methods, including the user-space and polling mode drivers [13, 29, 35, 42, 52, 54, 61], into the software AVS and achieved the performance of 10 Gbps/1.5 Mpps for each single CPU core. Despite these improvements, the software AVS still falls short of the line rate and ~100 Gbps bandwidth in modern Network Interface Controllers (NICs), which has been warned in [57]. This shortfall prompted us to use SmartNICs with hardware accelerators, such as Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs), for efficiently offloading "match-action" tasks [30, 50].

To tackle skewed network traffic distribution in cloud data centers and accelerate most of the popular traffic [27, 55], we researched existing works and found that some mainstream solutions divide packet forwarding into two separate data paths: a software data path running the whole vSwitch, and a hardware acceleration path acting as a forwarding cache [5, 6, 12, 37]. In this paper, we refer to this design as ***"Sep-path"***. Due to its minimal intrusiveness and straightforward integration, we chose the "Sep-path" for accelerating AVS with no major changes to the existing software framework.

However, the deployment of the "Sep-path" architecture in Alibaba Cloud has revealed several challenges for AVS, in addition to its advantages in handling popular traffic. These issues arising from large-scale deployment can be categorized into three types:

- First, the uncertainty in the selection of data paths for tenant traffic leads to unpredictable performance. The "match-action" tasks in AVS cannot always be implemented in the hardware data path due to the hardware limitations. In at least 10% of cases where "match-action" tasks are not suitable to implement in hardware accelerators, the performance is compromised and Service Level Agreements (SLAs) may not be guaranteed.
- Second, the long development life cycle of hardware accelerators hinders the evolution of AVS. The "Sep-path" architecture necessitates the concurrent design and implementation of "match-action" tasks in both software and hardware accelerators to accommodate emerging cloud network services, decelerating the iteration velocity of AVS [39, 44, 64].
- Last but not least, the "Sep-path" architecture increases maintenance costs due to the introduced hardware acceleration data path. Statistical analysis in Alibaba Cloud indicates that 65% of AVS deployment issues stem from hardware and software-hardware interactions, further exacerbating the prolonged development cycle issue.

In this paper, we introduce *Triton*, a flexible hardware offloading architecture for AVS, designed to address these limitations. In contrast to the "Sep-path" architecture, *Triton* performs all packet

★Co-first authors □Co-corresponding authors

processing in a *single* pipeline divided into three stages: the Hardware *Pre-Processor*, the Software Processing, and the Hardware *Post-Processor*. To derive a reasonable division of functionality, we measured the CPU usage of AVS under a typical workload (§4.1), and chose the following distribution: the *Pre-Processor* and *Post-Processor* stages leverage hardware to implement generic but time-consuming tasks (*e.g.*, parsing, fragmentation, and encryption) for accelerating the Software Processing stage which runs the highly flexible "match-action" tasks (§4.2).

To maximize the potential of hardware acceleration and mitigate bottlenecks in *Triton*, we adopted a series of hardware-software co-designs (§4.3). The techniques include a flow-based packet aggregation mechanism in hardware and vector packet processing in software for improved packet rate (§5.1), as well as support for jumbo frames and a novel header-payload slicing scheme for enhanced bandwidth (§5.2). Our evaluations demonstrate that these co-designs enable *Triton* to achieve comparable packet rate and bandwidth to the hardware data path in "Sep-path", while preserving software flexibility (§7). The insights and lessons learned from deploying *Triton* in Alibaba Cloud may offer valuable guidance for the academic community and other cloud vendors (§8).

The detailed contributions can be summarized as follows:

- First, we introduce the experience in accelerating AVS in Alibaba Cloud and present *Triton*, a flexible hardware offloading architecture. Differing from our prior solution based on the "Sep-path" architecture, *Triton* aims to balance performance and flexibility by unifying the data paths, and modeling to selectively offload generic network tasks to hardware accelerators.
- Second, we fully explore hardware and software co-designs to address potential performance issues in the software part of *Triton*, including vector packet processing and header-payload slicing to improve packet rate and bandwidth.
- Third, we deploy *Triton* in Alibaba Cloud to accelerate AVS and conduct real-world evaluations. The results demonstrate that *Triton* also achieves comparable performance acceleration to our prior solution based on the "Sep-path" architecture besides the advantage in predictable performance and iteration velocity. With the hardware-software co-designs, *Triton* improves AVS connection establishment rate by 72%, with only 2 μs latency increased than the hardware data path in "Sep-path".

2 BACKGROUND AND MOTIVATION

In this section, we first introduce AVS and its software framework evolution from kernel space (version 1.0 and 2.0) to user space (version 3.0). Based on AVS 3.0, we developed a hardware offloading solution under the widely used "Sep-path" architecture. Unfortunately, we found that it brings issues including unpredictable performance and limited flexibility under large-scale operations.

2.1 Apsara vSwitch (AVS)

The AVS, integral to the Achelous network virtualization platform [63], is tasked with providing network connectivity and management for instances including VMs, bare metals, and containers. Central to its functionality, AVS efficiently matches incoming packets with a series of predefined policy tables and executes corresponding actions, which are vital to the forwarding performance.

Distinct from open-source vSwitches, which mainly focus on stateless forwarding and portability, AVS is tailored for stateful cloud services. First, without the demand for cross-platform deployment, AVS can be customized for high performance in both hardware and software. For example, the flow table in AVS is customized for many stateful services, and we proposed optimizations such as the "session" structure (see Sec. 2.2). Second, under the pressure of managing large-scale table entries in the cloud network, we made a more flexible decoupling between the data plane and the control plane in AVS for rapid network convergence [63]. Third, AVS relies on stronger operation and maintenance capabilities, including statistics, diagnosis, and visualization. Among them, visualization also requires us to provide tenants with a variety of tools like Traffic Mirroring [11, 19] and Flowlog [8, 16]. Last but not least, programming flexibility and hot upgrade capabilities are required for AVS to support emerging new services in the rapidly evolving cloud infrastructure [1, 39, 44, 46, 47, 53, 64].

2.2 The Evolution of AVS Acceleration

Kernel modules. The development of AVS began more than ten years ago when Alibaba Cloud was first established. The earliest version of AVS 1.0 was not an independent process, but a collection of system services. In AVS 1.0, we made full use of the existing Netfilter modules [24] in the Linux kernel to implement various network functions. For example, we used Traffic Control [23] to implement the Quality of Service (QoS) function and used iptables [22] to implement network security groups. To solve the performance and management issues in these distributed modules and reduce dependence on the Linux kernel, we developed AVS 2.0 which is similar to the current vSwitches. It contains two processes: the packet forwarding process in the kernel space and the daemon process for management in the user space. In the packet forwarding process, AVS 2.0 abstracts the service layer and public layer to flexibly iterate while improving forwarding performance.

User space acceleration. To minimize the overhead in kernel space packet processing and eliminate the dependency on Linux kernel versions, AVS 3.0 adopted the Data Plane Development Kit (DPDK) [13] as a user-space acceleration solution. Fig. 1 illustrates the implementation of the Fast Path and Slow Path designs in AVS 3.0, aimed at boosting the packet forwarding rate. Central to the Fast Path design is the "session" structure, which comprises a pair of bidirectional flow table entries and their associated states. Consequently, when packets are processed via the Fast Path, their flow entries are efficiently indexed into the "session" for stateful processing, eliminating a separate module for connection tracking. This session-based architecture significantly accelerates network performance, particularly for stateful services like Network Address Translation (NAT) and Load Balance (LB). However, the AVS 3.0's performance of 10 Gbps/1.5 Mpps per CPU core still falls short of matching the line rate of modern NICs. To make matters worse, we found that deploying AVS on hosts also led to resource contention with tenant VMs, notably in bus and cache resources. These factors motivate us to introduce hardware outside the host to offload AVS.

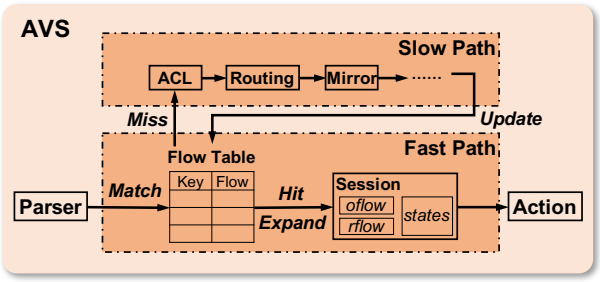


Figure 1: The architecture of AVS. We flexibly expand the matching tables on the Slow Path and accelerate stateful processing through "session" on the Fast Path.

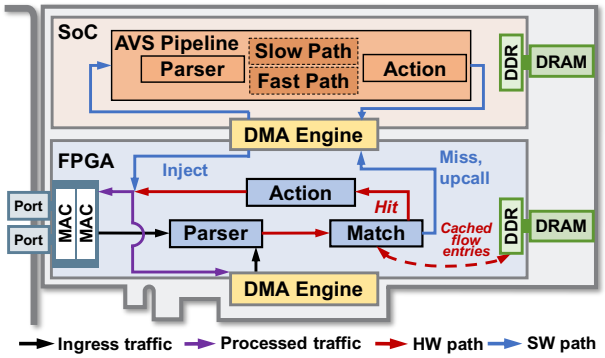


Figure 2: "Sep-path" hardware offload architecture for AVS. We have deployed it on our self-developed SmartNIC and operated it for years.

Software offloading on the System on Chip (SoC) based SmartNIC. Typically, there are two extreme directions on accelerating vSwitch in SmartNIC: hardware full offloading, and software implementation on SoC CPU cores. The hardware full offloading approaches utilize ASIC/FPGA to fully embed the packet processing pipeline in hardware, achieving high performance at the expense of flexibility. In contrast, the software-based approaches operate the vSwitch entirely on SmartNIC's CPU cores, prioritizing flexibility over raw performance. But our experimental deployment with this approach adopted on AVS revealed that while it successfully eliminates packet copying between host and VM memory (*i.e.*, between the front-end and back-end of the virtio [56] device), it does not enhance much overall performance. The limitation primarily stems from the poor performance of SmartNIC's CPU cores compared to host CPU cores, a consequence of power limitation.

"Sep-path" hardware offloading architecture. To compromise between the two extreme directions for accelerating AVS, we chose to implement an offloading architecture similar to the VFP acceleration solution [37] and Mellanox OVS offloading method [6]. We named it as ***"Sep-path"***, because it manages two separate data paths: the hardware data path acts as a flow cache for acceleration; the software data path runs the whole vSwitch on SoC. The AVS acceleration solution based on "Sep-path" is shown in Fig. 2, which is implemented on our internally developed SmartNIC. The SmartNIC splits into software (SoC) and hardware (FPGA) components, linked by 2×8 PCIe 4.0 channels, with packet forwarding logic executed across both. The hardware path, distinguished by offloading popular flow table entries, accelerates the cached flows, whereas uncached flows are processed by the software.

The "Sep-path" architecture introduces an additional hardware forwarding path and necessitates minimal modifications to the existing AVS software framework, which allows us to deploy it in a short time. *However, it is also the two forwarding paths that separate the originally unified performance metric and programmability in AVS, presenting challenges in large-scale cloud deployment.*

2.3 Deployment Issues

After several years of deployment, we observed that the "Sep-path" architecture suffered from the following issues:

Unpredictable performance. Our deployment experience and analysis challenge the notion that average data accurately reflects the "Sep-path" architecture's efficiency. As shown in Table 1, we evaluate the Traffic Offload Ratio (TOR, calculated by offloaded traffic bytes/all traffic bytes) in 4 typical regions of Alibaba Cloud, and find inconsistencies. For example, even in Region C with the highest average TOR and only 1.9% of the host has TOR less than 50%, 25.5% of tenant VMs failed to benefit significantly from hardware offloading because half of their traffic was still processed via the software data path. In Region D, nearly half of the total VMs have a TOR below 50% while the average TOR in this region is 81%. That suggests only a small proportion of tenants with long connections and heavy traffic contribute the main TOR in cloud data centers, while the traffic of most tenants remains unoffloadable due to the short connection and hardware resource constraints. A typical example of hardware resource constraints is that the hardware data path can only afford to store RTTs for tens of thousands of flows for Flowlog product, and the excessive flows must go through the software data path. Consequently, VM network performance in these scenarios is constrained to the capacity of software forwarding, impacting the ability to ensure SLAs. Furthermore, excessive unoffloaded traffic on a single host can lead to CPU resource contention, adversely affecting the tenant experience.

Decreased iteration velocity. Iteration is vital on cloud platforms for supporting new services and resolving bugs [39, 44, 64]. However, the unpredictability of future services, protocols, and actions complicates AVS's adaptability, requiring direct source code modifications instead of simpler SDN-style orchestration. Over the past three years, we have integrated more than 20 new features into AVS, including one requiring parser modification, three requiring adjustments to match fields (such as adding instance IDs or new protocol headers), and seven requiring new "actions". The "Sep-path" architecture amplifies the workload of design and development, demanding parallel efforts on software and hardware data paths. That led to a reduced iteration velocity, falling from 4-5 releases per year to approximately 2-3 releases per year.

High maintenance costs. The deployment of the "Sep-path" architecture for AVS acceleration has notably reduced maintenance efficiency, particularly in terms of increased time required for bug identification. For instance, in cases of packet loss, it is necessary to first ascertain the data path (hardware or software) where the

Regions	Average TOR sum(offload)/sum(all)	Host level distribution (TOR < 50%)	Host level distribution (TOR < 90%)	VM level distribution (TOR < 50%)	VM level distribution (TOR < 90%)
Region A	90%	5.7%	29.4%	39.8%	63.3%
Region B	87%	7.9%	42.3%	37.3%	63.7%
Region C	95%	1.9%	15.8%	25.5%	50.3%
Region D	81%	7%	45%	43%	66%

Table 1: The Traffic Offload Ratio (TOR) distribution at Host and VM level in typical regions.

packet loss occurred, followed by detailed analysis. Hardware path troubleshooting is especially time-consuming, as it largely relies on reading values in registers without any useful run-time debugging tools. In certain scenarios, this often leads to extended periods spent analyzing discrepancies in flow cache entries and sessions between hardware and software. The operational statistics from nearly two years of deployment show that of all the functional bugs, design flaws, and product limitations, 25% are caused by hardware, 40% are caused by software-hardware interaction (due to the complex synchronization between two data paths), and only 35% are caused by only AVS itself. Moreover, the issues involving hardware usually take more time to resolve.

It is important to clarify that the deployment challenges encountered with the "Sep-path" architecture are not inherently due to its design, which involves two separate forwarding paths. From our perspective, the "Sep-path" represents a common optimization strategy, facilitating the acceleration of popular traffic with minimal modifications. *However, the disparity in performance and programmability between heterogeneous hardware platforms (i.e. CPU and FPGA) presents a significant gap. This gap renders the "Sep-path" architecture less suitable for rapidly iterative cloud vSwitches.*

3 TRITON DESIGN OVERVIEW

To reconcile the inconsistent performance and programming capabilities of the two separate data paths, we are moving towards unifying them and developing *Triton*, the new generation of hardware offloading architecture for AVS. The biggest improvement in *Triton* is that the packets are processed serially on both hardware and software, and the workloads are reasonably distributed among them. That allows predictable performance and programming flexibility to support emerging services.

3.1 Design Overview

As illustrated in Fig. 3, the *Triton* architecture processes all packets serially through both hardware modules and software components. The primary distinction of *Triton* from the prior "Sep-path" solution lies in its unified data path. This architectural shift offers several benefits. First, it promotes predictable performance, which is crucial for guaranteeing the SLAs for tenants. Second, it significantly reduces both development and operation costs.

As packet processing transitions from parallel to serial processing across software and hardware in the *Triton* architecture, there is a consequential shift in workload distribution. Unlike the "Sep-path", where workload distribution depended on whether the "match-action" can be offloaded, *Triton* allows for more granular distribution at different stages of packet processing. That enables precise workload modeling for packet forwarding, thereby assigning flexible operations to the software, and fixed yet time-consuming

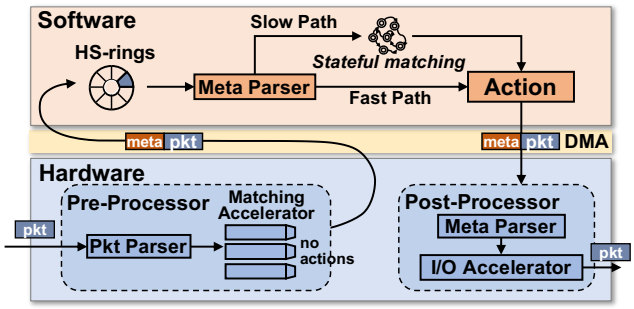


Figure 3: Triton architecture overview. Generic packet processing workloads are offloaded to hardware for acceleration, while the flexible logic is left to software.

tasks to the hardware. Consequently, *Triton* will be able to strike a balance between performance and flexibility, effectively addressing the limitations outlined in Sec. 2.3.

To realize the balance in *Triton*, we abstract the data path into three parts: Hardware *Pre-Processor*, Software Processing, and Hardware *Post-Processor* (see Fig. 3). The packet processing sequence in *Triton* is as follows. Initially, in the *Pre-Processor* stage, the hardware efficiently handles I/O offloading tasks, such as TCP Segment Offload (TSO), UDP Fragment Offload (UFO), and packet header parsing. Subsequently, a matching accelerator on hardware matches the packet, which is aimed to speed up the matching stage on the software side. As packets pass through the *Pre-Processor*, they are upcalled to the software along with metadata containing intermediate processing results, which enables AVS to speed up software "match-action". After Software Processing, packets are redirected to the hardware *Post-Processor* via Direct Memory Access (DMA) for final I/O operations, including encryption and egress. The detailed workload distribution in *Triton* will be demonstrated in Sec. 4.

3.2 The Advantages in Triton

Predictable performance. The unified data path in *Triton* offers predictable network performance with precise upper and lower performance limits, thereby ensuring compliance with performance SLAs. Unlike the significant performance gap between the software path and hardware path in the previous "Sep-path" architecture, the gap between the Fast Path and Slow Path within the software AVS is less consequential. The difference stems from the fact that path selection in the latter is influenced by tenant behavior rather than whether the workload can be offloaded. For example, packets establishing new connections consume more CPU cycles on the Slow Path, while the packets of established connections pass faster

over the Fast Path. This fact is similar in the protocol stack processing within the guest OS, which is more likely to be a bottleneck than AVS. Consequently, the unification of software and hardware paths under *Triton* aligns VM network performance metrics more closely with tenant expectations.

Flexibility and iteration efficiency. In *Triton*, the distribution of workloads between software and hardware is well-designed to achieve a balance between performance and flexibility. In Sec. 4, we will illustrate how we model and effectively offload generic logic to the hardware, while maintaining the programming flexibility in software. This approach accelerates the iteration, as additional functions can be seamlessly added to the software without changing static hardware modules.

Maintainability with reduced operational costs. First, *Triton* minimizes redundant resource utilization, resulting in resource savings on the SmartNIC. These saved resources can be allocated to other hypervisor services, such as storage. Additionally, *Triton* assigns flexible and dynamic workloads to software, enhancing the debugging and troubleshooting capabilities. Consequently, operational efficiency can be improved through methods like full-link packet capture and dynamic code replacement. A comprehensive comparison of operational tools between *Triton* and "Sep-path" can be seen in Sec. 7.

Hardware acceleration without sacrificing flexibility. In the software-based offloading architecture, hardware primarily offers Rx and Tx capabilities, with packet processing being handled entirely on the SoC cores. While this approach reduces I/O overhead, it often faces performance issues. In contrast, *Triton* introduces hardware modules that facilitate the integration of specific packet processing stages, enhancing acceleration. This integration allows *Triton* to offload as many time-consuming tasks (like parsing) as possible to hardware for speedup without sacrificing flexibility.

Low cost for implementation. The cost associated with migrating from "Sep-path" to *Triton* is reasonable. Our implementation, as discussed in Sec. 6, shows that migration can be accomplished by eliminating redundant logic in hardware (the FPGA units) and introducing less than 2000 lines of code to AVS.

4 TRITON PACKET PIPELINE

In this section, we focus on optimizing *Triton* by effectively modeling the AVS packet processing tasks, and distributing the workload between software and hardware. We first present the AVS workload model, then describe *Triton*'s packet pipeline design, and finally discuss the associated performance issues.

4.1 AVS Workload Model

We model the AVS packet processing pipeline, and divide it into three stages: parsing, matching, and action execution.

Parsing stage. Upon receiving a packet, the AVS performs various operations such as validation, header parsing, and extraction of matching fields. During this stage, latency is introduced as the CPU waits for multiple memory accesses to retrieve fields from multiple header layers. As indicated by the *perf* results in Table 2, these operations account for 27.36% of the CPU usage dedicated to forwarding.

Stage	Cost (CPU usage)	Workload Distribution
Parsing	27.36%	Hardware
Matching	11.2%	Software & Hardware assisted
Action	24.32%	Software & Hardware assisted
Driver	29.85%	Software & Hardware assisted
Statistics	7.17%	Software

Table 2: The CPU usage of each stage during packet processing in software AVS, and the ideal workload distribution in Triton.

Matching stage. Following initial parsing, the AVS matches relevant fields to identify the corresponding flow entry. The pipeline encompasses both a Slow Path, typically for the first packet, and a Fast Path aimed at accelerating subsequent packets. It is important to recognize that the static rule lookup alone may be insufficient for stateful packet processing. For example, stateful ACL requires the acceptance of all reply packets once the request packets are dispatched; in certain scenarios, recording the IP/MAC of a physical host as the next hop for reply packets is necessary. It can be seen that matching is quite a time-consuming and flexible task. Our findings indicate that even for long-lived connections primarily handled on the Fast Path, hash lookup accounts for approximately 11.2% of CPU consumption.

Action execution stage. The action execution stage in AVS, characterized by flexibility, executes all the actions gained from the matching stage, including VXLAN encapsulation, NAT, and fragmentation [43]. It adapts to new services by expanding its action set. Performance tests (using *perf*) reveal that the basic overlay network forwarding actions consume about 24.32% of CPU resources.

Beyond the three stages, two additional workloads also impact AVS CPU usage. NIC driver processing, exemplified by the virtio driver, constitutes 29.85% of CPU usage. This is attributed to its role in packet transmission and reception, including comprehensive checksum calculations. Notably, checksumming (*e.g.*, in L3 and L4 headers) is required for both the overlay and underlay protocol headers. Additionally, operational code execution accounts for 7.17% of the total CPU consumption.

4.2 Packet Pipeline in Triton

Based on the model above, we meticulously distribute workload and design the packet processing pipeline in *Triton*.

Parsing (on hardware). As depicted in Fig. 3, our design incorporates a dedicated *Pre-Processor* module in hardware to offload the entire parsing stage. To ensure efficient transmission of parsing results to the software, we have devised a metadata structure that stores the intermediate outcomes. When a packet is received, the *Pre-Processor* module validates it and extracts the five-tuple in various header layers to facilitate packet matching. The extracted intermediates are then stored within the metadata structure. Once the parsing is completed, the metadata structure will be positioned

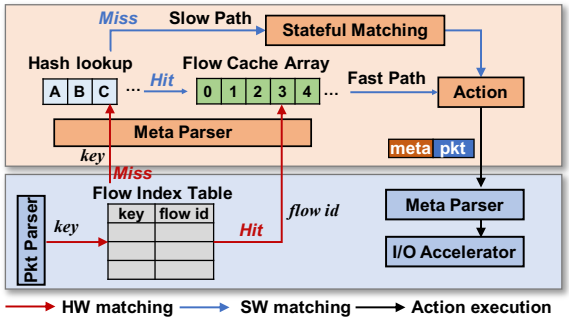


Figure 4: Hardware-assisted software packet matching.

ahead of the original packet to subsequently be passed on through PCIe channels to the software.

Matching (on hardware & software). In matching stage, our objective is to leverage hardware modules to mitigate the hash lookups on the software Fast Path.

As shown in Fig. 4, we introduce a "Flow Index Table" in the *Pre-Processor* module for acceleration. This table does not store the entire flow entry which contains matching fields and actions. Instead, it serves as a mapping between the key computed by five-tuple hash, and the respective "flow id". The "flow id" acts as an index within the "Flow Cache Array" (as illustrated in Fig. 4) within the software AVS.

Once the packet has acquired the corresponding "flow id" through a lookup in the "Flow Index Table", this "flow id" will be stored in the metadata and transmitted to the software. Furthermore, irrespective of whether the packet is matched in the "Flow Index Table", the *Pre-Processor* will write the packet along with the metadata into the HS-rings. The HS-rings represent the queues located in SoC DRAM that facilitate interaction between the hardware and software, as depicted in Fig. 3. The software will undertake the necessary subsequent matching operations.

Upon the packet arrives at the software AVS, the CPU cores on the SoC will attempt to process the packet through the Fast Path, as shown in Fig. 4. The CPU retrieves the `flowid` from the metadata fields and directly accesses the corresponding flow entry. However, if the hardware matching fails and the `flowid` is empty, a hash lookup is required to locate the relevant flow entry. If the packet fails to find the flow entry on the Fast Path, it will undergo processing through a series of matching tables and even stateful processing on the Slow Path. Following successful matching in Slow Path, the resulting actions are consolidated into a list. To accelerate the processing of subsequent packets within the same flow, a flow entry is generated on the Fast Path, encompassing the hash key, five-tuple, and action list.

Action execution (on software, I/O left for hardware). After matching, the CPU undertakes traversal of the action list within the flow entry and executes various flexible actions such as VXLAN encapsulation, NAT, QoS, and others. Conversely, the hardware (*Post-Processor*) handles I/O-intensive actions, such as fragmentation and checksumming. This approach effectively reduces the CPU overhead associated with NIC driver checksumming (8% for physical NICs and 4% for vNICs). By primarily relying on software for action execution, *Triton* offers enhanced flexibility and scalability. For instance, the system easily accommodates the incorporation of complex actions to support jumbo frames (see Sec. 5.2).

Triton's packet processing pipeline also offers the additional advantage of simplifying updates and synchronization. This is achieved by eliminating the flow cache in the hardware, thereby eliminating the requirement for the software to synchronize flow information with the hardware. Moreover, since every packet undergoes processing in both the hardware and software components, updates to the "Flow Index Table" can be seamlessly executed through instructions embedded within the metadata. As a result, the process of updating and synchronizing becomes more streamlined within *Triton*'s packet processing pipeline.

4.3 Potential Performance Issues

While *Triton*'s unified data path design provides a good balance between predictable performance and flexibility, it is essential to recognize that incorporating the software onto the per-packet data path may give rise to potential performance issues. These issues encompass the following aspects:

Packet rate issue. In *Triton*, the processing of each packet involves time-consuming software operations, including packet matching and action execution. The overall number of packets that can be processed per second is constrained by the processing capability of the CPU. As a result, when compared to the "Sep-path" solution, *Triton* may potentially diminish the benefits of hardware acceleration in terms of packet rate performance (measured in PPS, the abbreviation of Packets Per Second).

Bandwidth issue. As each packet necessitates movement between hardware and software, the available PCIe bandwidth might be exhausted. In the depicted SmartNIC, the transfer of packets between hardware modules and the SoC is achieved through DMA operations (see Fig. 2). In the case of *Triton*, every packet is DMAed from the blue hardware module to the orange SoC for processing, and subsequently DMAed back to the hardware module. These two DMA operations occur on the same PCIe bus, resulting in the halving of available bandwidth.

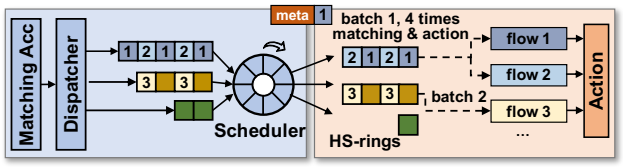
It is worth noting that these challenges pose formidable obstacles in traditional software AVS architectures like software-based offloading solutions on SoC. However, in *Triton*, these issues can be effectively tackled through the utilization of hardware acceleration modules. Through the optimizations highlighted in Sec. 5, we demonstrate that *Triton* can achieve high PPS and bandwidth performance comparable to the hardware forwarding in "Sep-path".

5 PERFORMANCE OPTIMIZATION

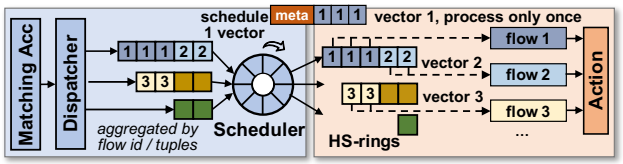
In this section, we present the technological innovations in *Triton* to effectively address the potential performance issues, including packet rate performance and bandwidth limitations.

5.1 Packet Rate Improvement

Through observation and analysis of software processing, we found that the widely used batch processing optimization does not allow the CPU to process packets efficiently. When processing a batch of



(a) Batch Packet Processing



(b) Vector Packet Processing

Figure 5: The comparison of batch processing and vector processing in Triton.

packets (*e.g.* 32 or 64 packets), the CPU will perform the "match-action" for each packet in sequence (see Fig. 5(a)). So it can be seen that the batch processing optimization cannot reduce the overall times of packet matching and the instruction cache miss rate. To this end, we propose a flow-based packet aggregation mechanism that uses hardware to aggregate packets belonging to the same flow into a vector, so that software can do Vector Packet Processing (VPP, which only requires one matching for a vector) to save CPU cycles. Compared to other existing software-based vector packet processing solutions [31, 42], leveraging hardware for aggregating packets of the same flow can maximize the benefits on packet rate.

Flow-based packet aggregation. As shown in Fig. 5(b), the *Pre-Processor* module in *Triton* employs a flow-based packet aggregation mechanism to group packets belonging to the same flow into a *vector*. The aggregation takes place after the packet passes parser and matching accelerator. When a packet matches an entry in the "Flow Index Table", it will be aggregated based on the "flow id". Conversely, if a packet fails to match, it will be aggregated based on the five tuples. Subsequently, the vector of packets will be transferred to the HS-ring, with the vector size indicated in the metadata of the first packet. To minimize latency, packet aggregation module follows the best effort principle and its efficient implementation will be discussed in Sec. 8.

Vector packet processing (VPP). In *Triton*, the software packet processing has also transitioned to vector-based granularity. As shown in Fig. 5(b), upon receiving a packet from the HS-ring, the CPU retrieves the vector size from the metadata. Subsequently, it traverses backward until all packets within a vector are fetched. For each vector, it only requires one matching operation to retrieve the flow entry and corresponding action list. This VPP approach significantly reduces the times of packet matching within a single flow. In comparison to traditional batch processing optimization (Fig. 5(a)), this technique greatly enhances packet prefetching efficiency and increases cache hit rates. The effectiveness of this approach will be demonstrated in Sec. 7.2.

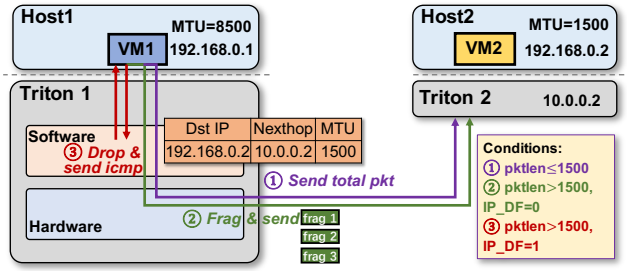


Figure 6: Triton ensures connectivity in multi-MTU scenarios. VM2 acts as a stock VM that only supports 1500 MTU.

5.2 Bandwidth Improvement

Triton achieves bandwidth improvement through two fundamental principles: increasing the packet payload and minimizing unnecessary data movement. In this subsection, we introduce two techniques: jumbo frames support in cloud data centers and header-payload slicing.

Jumbo frames support. Supporting jumbo frames is crucial for enhancing bandwidth and transmission efficiency. It increases the effective payload per packet, reducing performance pressure on forwarding components. For example, using 8500-byte jumbo frames instead of 1500-byte packets can increase payload by 14% and reduce demand for packet rate by up to 82%.

However, supporting jumbo frames in cloud-scale data centers presents more compatibility challenges than supporting it within only a specific cluster. There are a number of stock VMs and out-of-date hardware devices that do not support such a big Maximum Transmission Unit (MTU), but have the demand to communicate with VMs that use 8500 MTU. To make things worse, most hardware switches do not support fragment and Path MTU Discovery (PMTUD) mechanisms. That requires AVS to negotiate MTU and ensure network connectivity for VMs.

To tackle this, we introduces path MTU and new actions to AVS for ensuring network connectivity in multi-MTU scenarios. The controller attaches the path MTU when issuing routing entries to AVS, allowing awareness of the maximum acceptable MTU to the destination. Then AVS employs three new actions, as following RFC [2, 3, 32] standards, for handling oversized packets (see Fig. 6). For a packet that is smaller than the path MTU, the AVS will forward the packet as usual. But when the packet is bigger than the path MTU, there will be two different treatments according to the standards. If the Don't Fragment (DF) field in IP header is set to 1, the packet should be dropped and an ICMP message containing path MTU will be sent to the source VM to reduce packet length. This kind of action is complex and costs too much to generate a new packet in hardware, so we implement it in software AVS. But if the DF field is 0, the packet should be fragmented and sent out. The action is fixed and I/O related, thus is more suitable to be implemented in *Post-Processor* for efficiency.

Header-payload slicing (HPS). To mitigate PCIe bandwidth usage caused by needless data movement between software and hardware, *Triton* presents a technique called Header-Payload Slicing (HPS). This design is rooted in two key observations: the primary focus

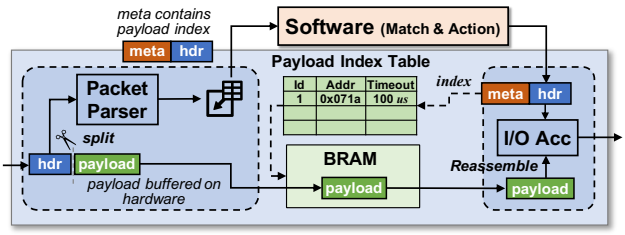


Figure 7: HPS reducing PCIe bandwidth occupation during packet movement.

of packet processing in the AVS mainly lies in packet headers, and the payload typically accounts for the majority of the bandwidth in a packet.

The workflow of HPS is depicted in Fig. 7. The *Pre-Processor* module divides the packet into header and payload segment. Once the *Pre-Processor* completes matching acceleration, only the packet header and associated metadata are delivered to the software via the HS-rings. While the packet header undergoes processing through the software pipeline, the payload is stored in the BRAM until the header completes the action execution in the software and is sent back for reassembly. That significantly reduces the PCIe bandwidth consumption during DMA operations between the software and hardware. For example, when forwarding an 8500-byte packet, HPS can save approximately 97% of the PCIe bandwidth.

Following the completion of software processing and the transmission of the packet header back to the hardware, *Triton* requires an efficient mechanism to locate the corresponding payload and reconstruct the packet. To accomplish this, we employ a "Payload Index Table" to manage the mapping between headers and payloads. During packet division, the *Pre-Processor* attaches the payload index to the metadata. Upon the header is return from the software, the *Post-Processor* module utilizes the index in the metadata to locate the corresponding payload in the "Payload Index Table" and reassembles it into a complete packet for transmission.

In actual deployment, we found the biggest problem in HPS is that the BRAM may be exhausted if the buffered payloads are not reassembled in time. For example, when the software pipeline processes too slow or encounters an exception, headers are not returned on time and there are not enough buffers to store incoming packets. To this end, we introduced timeout and version management mechanisms to solve this problem. Since the software only consumes a few microseconds to process a batch packets, the timeout value of each payload needs to be set small enough, such as 100us. Then, for payload buffers that are reused for timeout, we can avoid misuse by comparing versions when reassembling.

6 IMPLEMENTATION

In this paper, we compare the "Sep-path" version of AVS offloading architecture with *Triton* version. Both are implemented based on our internally developed SmartNIC (also referred to as CIPU [10]), which integrates FPGA hardware logic units with Intel x86 CPU cores on the SoC. The transition from the "Sep-path" to *Triton* involved adding less than 2000 lines of code to AVS and reducing redundant FPGA code on the hardware. The manpower investment for this transition was minimal, requiring just a few engineers for less than a month to develop the prototype. *Triton* optimizes resource usage, employing only 57K Look Up Tables (LUTs) and 6.28 MB buffers for its *Pre-Processor* and *Post-Processor*, which results in a 136K LUTs reduction compared to the original "Sep-path" architecture. The saved resources can be used to exchange for additional SoC CPU cores for higher network performance or used to serve other hypervisors like storage and computing. Last but not least, although our current implementation is based on FPGA, *Triton*'s hardware modules could potentially be implemented by using newer technologies like eASIC [14] for improved performance and cost-effectiveness.

7 EVALUATION

In this section, we initially evaluate the enhancements of *Triton* in relation to the prior "Sep-path" architecture, focusing on AVS performance, flexibility, and operational efficiency, with identical hardware conditions. Subsequently, we assess the effectiveness of the hardware acceleration techniques discussed in Sec. 5. Finally, we conduct application performance tests under varying traffic settings, including both long-lived and short connections.

7.1 Overall System Evaluation

First, we evaluate the performance, flexibility, and operational capability under *Triton* and our prior "Sep-path" architecture. The hardware costs under two architectures are equivalent: "Sep-path" uses 6 CPU cores and a hardware data path, while *Triton* uses less hardware resources and 8 CPU cores on the SoC (the 2 more CPU cores are gained by saving hardware resources).

Triton System Performance. Fig. 8 compares *Triton* and the "Sep-path" architecture regarding bandwidth, PPS, and Connections Per Second (CPS) performance. We use *iperf* [21], *sockperf* [26] and "CRR" mode in *netperf* [25] to test the bandwidth, PPS and CPS performance respectively. All these programs are run on multiple processes/threads to obtain the maximum forwarding performance of the whole system. Compared with the software path in "Sep-path", *Triton* increases the bandwidth and PPS by 2 times and 1.25 times respectively. Although there is still a small gap compared to 24 Mpps of the hardware path, 18 Mpps is sufficient to accelerate most of the tenants according to our operational statistics. The PPS performance in *Triton* can also be scaled up by increasing the number of CPU cores in the future. Regarding CPS, *Triton* exhibits a 72% improvement compared to "Sep-path". This is because the hardware path in "Sep-path" cannot accelerate the establishment of new connections, while *Triton*'s hardware-assisted design can enhance CPU efficiency.

Regarding latency, *Triton* introduces approximately 2.5μs of latency due to the per-packet interaction on HS-rings, as shown in Fig. 9. However, the impact on cloud services is negligible. That is because the latencies of applications like Redis [17] and Nginx [15] are all in the millisecond (see Sec. 7.3), and the bottleneck is in VM kernel processing.

Predictable Performance. We conduct tests for the route refresh scenario to measure the predictable performance. Both architectures initially support 2 million connections. We start to refresh the route table at 17 seconds to force all traffic upcalled to Slow Path

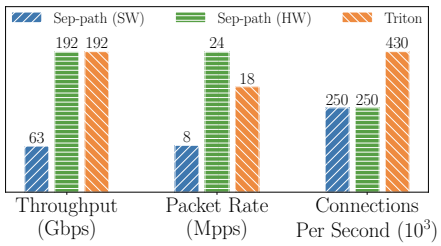


Figure 8: Overall performance

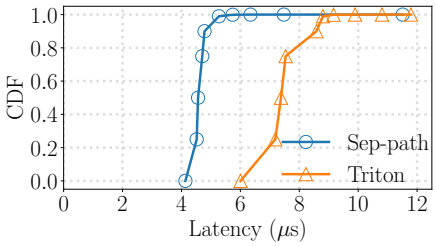


Figure 9: Latency overhead

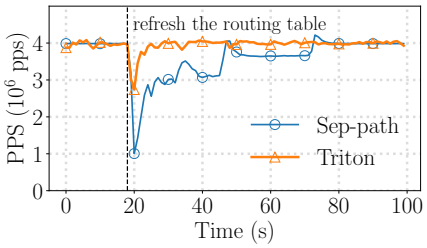


Figure 10: Predictable performance

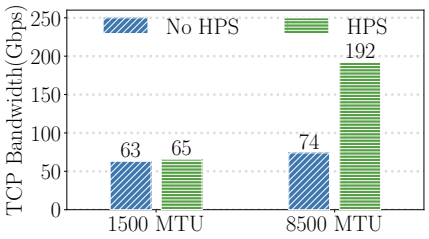


Figure 11: Bandwidth improved by HPS

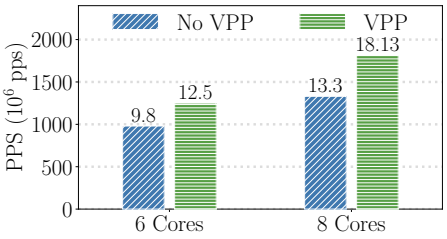


Figure 12: PPS improved by VPP

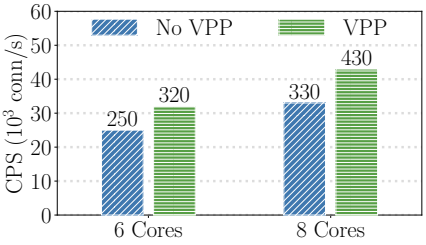


Figure 13: CPS improved by VPP

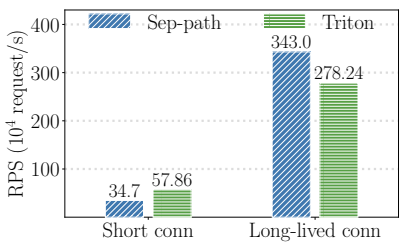


Figure 14: Nginx RPS Performance

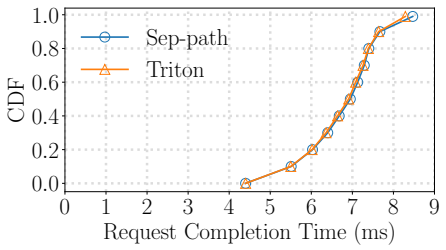


Figure 15: Nginx RCT (long conn)

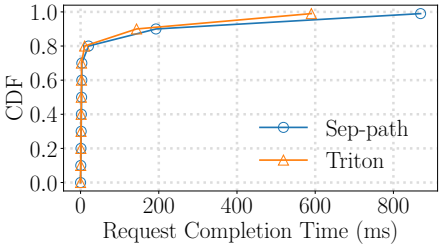


Figure 16: Nginx RCT (short conn)

for updating the flow cache. Fig. 10 illustrates the PPS in different architectures over 100 seconds. We observe a significant performance drop in the "Sep-path", approximately 75% lower than its initial performance (*i.e.*, the software/hardware gap), lasting for about 1 minute. At the minute, the CPU cores are busy forwarding traffic and issuing flow cache entries to hardware, bringing such a huge jitter. In contrast, *Triton*'s unified data path only experiences a 25% drop in performance (fast/slow path switch) within seconds. That indicates *Triton* is more capable of providing tenants with predictable network performance.

Flexibility. In terms of flexibility, *Triton* can easily support new features and rapidly iterate to new releases. Compared to "Sep-path", *Triton* only requires to modify software AVS with hardware modules unchanged for evolution. According to our release record, it significantly shortens the development life cycle by more than 50%. This result benefits from the fine-grained workload distribution, which enables *Triton* to support highly complex actions like the PMTUD shown in Sec. 5.2.

Operational efficiency. The unified data path in *Triton* makes troubleshooting easier. Firstly, by reducing synchronization, there are fewer compatibility bugs. Secondly, *Triton*'s primary and flexible workloads are implemented in software, which provides more powerful operational tools. As shown in table 3, *Triton* supports packet capture at each critical point, enables more detailed traffic statistics, and provides run-time debugging capabilities. For comparison, on the hardware path of "Sep-path", we lack these important debugging capabilities. We need to note that it has been a trend to adopt reliable overlay protocols and support multi-path transmission to avoid packet loss in cloud data centers [51, 60]. In this scenario, *Triton* provides a feasibility that the overlay protocol stack can be implemented on the per-packet software data path, which is difficult to achieve in the "Sep-path" architecture.

Operational tools	Sep-path	Triton
Pktcap points	Software only	Full-link
Traffic stats	Coarse-grained	vNIC-grained
Runtime debug	Software only	Full-link
Link failover	Unsupported	Multi-path

Table 3: Triton supports richer operational tools than Sep-path architecture.

7.2 Acceleration Techniques Evaluation

In this section, we detail how each innovation in Sec. 5 contributes to bandwidth and PPS improvement.

Bandwidth Improvement. Fig. 11 proves the effectiveness of jumbo frames and HPS in improving bandwidth. It can be seen that every single technique is limited in improving bandwidth because there is no significant improvement in DMA operations and the packet processing time of each packet. But when jumbo frames and HPS are both supported, *Triton* can achieve the same bandwidth as hardware forwarding. Our experiment proves that *Triton* can support bandwidth close to 200 Gbps. If the physical server supports multiple SmartNICs, the bandwidth can be further increased.

PPS/CPS Improvement. We conduct tests to compare the PPS and CPS performance before and after using VPP. Fig. 12 and 13 demonstrate that the flow-based packet aggregation and vector processing improve CPS and PPS by 27.6-36.3% (28% for 6 cores and 33% for 8 cores). That proves the hardware-assisted acceleration methods can significantly reduce CPU usage in software forwarding, thereby improving network performance in each dimension.

7.3 Application Performance

In this section, we deploy the real-world application to compare the performance of *Triton* and the "Sep-path" architectures. We select Nginx as the representative application because it is widely deployed on the cloud and can be used to simulate a variety of traffic characteristics. To validate the effects under different workloads, we choose long-lived and short-lived connections to represent two common workload types in production scenarios.

Nginx RPS Performance. Fig. 14 illustrates the Request-Per-Second (RPS) of Nginx with long connections and short-lived connections under *Triton* and "Sep-path" architectures. In the long connection test, *Triton* achieves an RPS of 2.78 million, which is 81.1% of the hardware path performance in "Sep-path" and sufficient for most tenants. In the short-lived connection test, *Triton* outperforms the "Sep-path" by 66.7%, reaching 578.6K RPS, indicating enhanced capabilities for establishing new connections and higher concurrent performance.

Nginx RCT Distribution. Fig. 15 and 16 show the Request Completion Time (RCT) results. In the case of long-lived connections, *Triton*'s latency is comparable with that of the hardware path in "Sep-path" (where the bottleneck lies in the VM kernel). In the case of short-lived connections, *Triton* significantly improves the long-tail latency. The p90 latency is reduced by 25.8% to 143.11 *ms*, and the p99 latency is reduced by 32.1% to 590.08 *ms*.

8 EXPERIENCE

We have deployed *Triton* on Alibaba Cloud for several years and gained much operational experience. In this section, we will first introduce some useful practices we explored in deploying *Triton*, then discuss how *Triton* can support reliable transmission and large-scale machine learning networks. In Sec. 8.2, we will demonstrate the principles we explored in the development and operation of AVS.

8.1 Deployment Experiences

Efficient implementation of flow-based packet aggregation in *Triton*. As shown in Sec. 5.1, vector-based scheduling is required in the flow-based packet aggregation strategy. In traditional designs, reordering and scheduling an uncertain number of packets will consume too many hardware resources and increase the system's complexity. Therefore, we tried to solve this problem by adding enough hardware queues to the *Pre-Processor* in our implementation. We used 1K hardware queues to store packets based on hash values calculated from five-tuple before scheduling packets to HS-rings. Ideally, the packets stored in each hardware queue should belong to the same flow (or to several flows under hash collision), eliminating the demand for packet reordering. Then each time, the scheduler selects up to 16 packets from each queue and DMAs them to HS-rings, then the software driver will count the number of packets in a vector. The DMA operation of each packet takes about 16 *ns* to complete, so the delay caused by scheduling 15 more additional packets is negligible.

Unnecessary packet loss avoidance with limited BRAM in *Triton*. Although we have adopted *Triton* with the techniques described in Sec. 5, there is still a risk of the SoC CPU cores becoming the bottleneck and causing packet loss due to buffer exhaustion. The bad news is that it will become more serious as the VMs can use higher-end CPU models. Under this trend, increasing hardware buffer size and using more powerful SoC CPUs cannot eliminate the risk. We believe it should be addressed by rate limiting to tenant VMs, and the mechanism should be as close to the source VM as possible to suppress the sending rate. So that the congestion and packet drop can be avoided on all the subsequent forwarding components.

Our approaches to handle congestion in the VM Tx and VM Rx directions are as follows. First of all, the *Pre-Processor* will determine whether the congestion will occur by monitoring the HS-ring water level in both directions. In the VM Tx direction, the *Pre-Processor* will slow down the rate of fetching packets from the corresponding VM's queues (there is a mapping relationship between the virtio queues and the HS-rings) to form back-pressure and reduce the sending rate in the guest OS. In the VM Rx direction, several mechanisms need to work together. We implemented a VM-level pre-classifier based on the mac address in the *Pre-Processor* to distinguish the "noisy neighbors" and do rate limiting to them, which can provide performance isolation for others. At the same time, the AVS on the destination host will notify the source AVS to form back-pressure to exact source VMs.

Postponing the TSO, UFO and checksumming operations on SmartNIC. Currently, the vNIC provides a rich series of offload operations, such as TSO, UFO, and checksumming for VMs. Naturally, they should be processed once the hardware module receives packets from the virtio queues (see ① in Fig. 17). However, in our deployment, we found that processing TSO and UFO at position ① in Fig. 17 cannot avoid fragment operations in subsequent processing. For example, when a packet is larger than the path MTU, it needs to be fragmented again. Therefore, we recommend postponing these I/O offload operations (such as TSO and UFO) to the *Post-Processor* (see ② in Fig. 17). On the one hand, it can relieve PPS pressure to

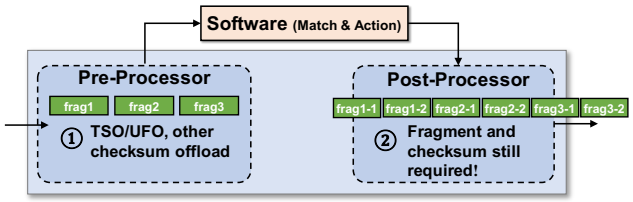


Figure 17: The reason why we should postpone TSO/UFO and other vNIC offload operations.

obtain greater bandwidth in software AVS because such big packets only require one "match-action" operation. On the other hand, the fragment and segment modules can be implemented centrally to save resources on hardware.

Enabling reliable transmission in *Triton*. Supporting multi-path and reliable transmission in cloud data centers has become a trend, but that requires new overlay protocols, such as SRD [60], solar [45] and falcon [20]. These new overlay protocols pose challenges to the vSwitch because the vSwitch should be able to switch paths in the network fabric and retransmit packets after packet loss. All these capabilities rely on the support of a specific protocol stack. However, under the "Sep-path" architecture, it is unrealistic for the hardware data path with independent forwarding capability to implement such complex protocol stack behavior. We find it is an opportunity for *Triton*, because the software AVS in the unified data path needs to process all packets, making it more suitable to deploy overlay protocol stack for reliable transmission. A feasible approach is to add a module for protocol stack processing in AVS, recording RTT and sequence for each packet, and triggering retransmission and path-switching behaviors when necessary.

Deploying *Triton* outside the Alibaba Cloud. *Triton* is a hardware offloading architecture decoupled from the customized processing logic of the AVS. The workload distribution method and optimizations can all be easily implemented on other vSwitches. For the hardware platform, *Triton* has no specific requirements for the hardware type on SmartNIC. Tofino, FPGA, and other ASICs are all enough to implement *Pre-Processor* and *Post-Processor* in *Triton*. Although we build it based on FPGA, it is not the most cost-effective approach. Alternatively, certain hardware like eASIC, can achieve higher performance and lower cost.

Supporting ~Tbps level bandwidth with *Triton* for large-scale machine learning. Irrespective of the hardware path or software path, the bandwidth is ultimately limited by the PCIe bus. With jumbo frames and HPS support, *Triton* can achieve up to 200 Gbps bandwidth on a single SmartNIC. Through the horizontal expansion of multiple SmartNICs, *Triton* is sufficient to support ~Tbps level bandwidth and higher PPS on a single physical server. Moreover, with the RDMA protocol stack being easily implemented in the software part of *Triton*, the requirements of large bandwidth and low latency for large-scale machine learning will be met.

8.2 Lessons Learned

Clarifying the boundaries of hardware capabilities. We hope this paper will draw attention to the gap between software and hardware. Not all workloads can be offloaded to hardware, and challenging the hardware boundaries may introduce endless operational issues. Taking the most common TSO and UFO functions as examples, some unusual packets such as IPv6 packets with extension headers and packets with padding data may not be suitable for hardware to fragment and segment. So we recommend clarifying the boundaries of hardware capabilities and always providing a failover method for rolling back to software when hardware fails to process the workload.

Establishing an integrated software/hardware testing system. To bridge the knowledge gap and information distortion between the software and hardware R&D teams, we formed a dedicated testing team to conduct comprehensive tests, including full-configured stress tests, performance tests, and functional tests, on the entire system. These tests have significantly enhanced the stability of AVS.

Live upgrade is the mean for serviceability. AVS needs live upgrade capability to rapidly iterate functional features. As the cloud vSwitch is a per-host component, the deployment task is immense. We perform live upgrades daily to fix bugs and support new features. Besides the synchronization of routing table entries, we also need to pay special attention to the network "downtime" during the switching procedure between the new and old AVS processes. For each physical or virtual interface's queue, there is only one AVS process that can Tx/Rx packets each time. To avoid traffic interruptions, we rely on traffic mirroring in the *Pre-Processor* to send packets to both old and new AVS processes for "match-action". In this way, no matter before or after the switch between the old and new AVS processes, there is a specific AVS process that forwards packets for the VMs. According to our operational records, the "downtime" of p999 VMs has been shortened to 100 *ms*.

Faster and more flexible resource scheduling on SmartNIC. Nowadays, cloud hypervisor services of network, storage and computing are all deployed on the SmartNIC (also known as DPU, IPU, CIPU, etc.), and the resources (such as CPU cores and memory) are always insufficient. But at the same time, we also observed that these hypervisor services rarely achieve peak usage simultaneously. So we implemented a dynamic resource allocation strategy for all the hypervisor services.

Pay attention to data visualization. We have implemented comprehensive data collection at various stages in both the hardware and software, encompassing logs, traffic statistics, and other relevant indicators. The collected data will be regularly updated in an analysis system to help us locate network issues and evolve our systems. For example, through these statistics, our monitoring system can provide a topology diagram of a pair of end-points in the cloud network at any certain moment, along with the status of each forwarding node in the network link. However, due to the limited hardware resources in the "Sep-path", we cannot complete all the data collection tasks in the hardware data path (*e.g.* collecting RTT, protocol, syn/rst/fin and other special statistics for each flow). The good news is that *Triton* brings the hopes of collecting fine-grained traffic statistics and developing more precise operational strategies.

Vendors and tenants can coordinate on operation and maintenance. In the host network, some failures or issues cannot be

avoided by just performing recovery operations on the vendor's side, such as the virtio queue hangs requiring the VM to reset the device. We provided tenants with as many vNIC statistics or events as possible for operation, which has been proven to make failover more efficient. At the technical level, we are promoting digitization capabilities through the virtio community to empower cloud vendors to disclose as many network statistics and events as possible to tenants inside the VMs based on virtio devices.

9 RELATED WORK

Hardware offloading architectures for vSwitch. Agilio CX [4] and Broadcom Stingray SmartNICs [7] can use SoC CPU cores to run software vSwitch, yet fail to sustain the growing bandwidth. Some hardware-only schemes [28, 40, 41, 49] focus on offloading the stateful packet processing pipeline to hardware. These schemes are resource-consuming and support very few actions. Moreover, their flexibility is limited by synthesis capability, making it inapplicable to the cloud. The "Sep-path" architecture represented by Accel-Net [37], ASAP^2 [6] and works in [5, 9, 12] employ a pipeline-based programmable FPGA or embedded switch (eSwitch) integrated into the NIC to implement virtual switching between virtual NICs. This hardware integration allows them to offload a significant portion of packet processing operations, resulting in high peak performance. However, this approach fails to guarantee SLA in short-lived connection scenarios and other unoffloadable scenarios. *Triton*, on the other hand, is an improvement upon the "Sep-path" architecture that ensures SLA through a unified data path, enabling AVS to achieve high performance while maintaining flexibility. This paper primarily focuses on the offloading acceleration architecture for cloud vSwitch, specifically the deployment form of packet data path within the SmartNIC. The discussion does not encompass the offloading strategies, offloading programming systems, associated domain-specific languages, or toolchains.

Application-specific hardware optimizations. The prior works have optimized the NIC for NFV. `nicmem` [48] and RIBOSOME [59] suggest that head-payload slicing can be used for shallow NFs to increase performance. *Triton* uses a similar mechanism and refines timeout mechanisms to avoid buffer exhaustion for differential processing speed issues. Backdraft [58] points out that when the number of NIC queues is substantial (over 1K), using user space drivers inevitably incurs polling overhead (100 CPU cycles per queue). Backdraft devises Doorbell Queues to solve the above problem, but it will also introduce latency. In *Triton*, we use hardware to aggregate a large number of virtio queues into the HS-rings (the number of HS-rings is pinned as the number of CPU cores) to reduce latency, which is more efficient. Reframer [38] shows the great benefit of aggregating flow-level packets to improve the memory locality in the packet ring buffer. We have proved in *Triton* how can this benefit be amplified by hardware assistance.

Versatile debugging and monitoring in Cloud Network. Previous work from cloud vendors illustrates the importance and complexity of vSwitch as an endpoint for O&M in cloud networks. Andromeda [33] has set up flexible components such as Tcpdump, Stats exporter, Packet tracer, Latency sampler, etc. in vSwitch software data path to get real-time network information. Large-scale cloud monitoring [34, 63, 64] requires vSwitch software to collect network information on end-to-end links with complex In-band Telemetry (INT) messages. Some expressive network monitoring tools [18, 62] support 100 Gbps traffic analysis on commodity hardware at the expense of 4~8 x86 cores. However, installing these components and mechanisms in the "Sep-path" architecture is hindered by constraints (e.g., hardware programming flexibility, hardware resources, and the performance of the NIC core). *Triton*, on the other hand, places flexible and dynamic workloads on software, enriching the debugging and troubleshooting methods with ease.

10 CONCLUSION AND FUTURE WORK

We present *Triton*, a flexible hardware offloading architecture for AVS in Alibaba Cloud. *Triton* stands out from our prior "Sep-path" architecture by striking a balance between performance, flexibility, and operability through the unified data path and comprehensive workload distribution model. We discuss the challenges faced during the deployment of *Triton* and the corresponding techniques employed to overcome them. The effectiveness of *Triton* has been validated within Alibaba Cloud. Compared to the "Sep-path", *Triton* achieves almost twofold increase in CPS and reduces application-level long-tail latency. Furthermore, *Triton* maintains unified performance metrics and flexibility. We anticipate that *Triton*'s expertise will be beneficial to the development of our next-generation SmartNICs, which are deeply customized for higher performance, low power consumption and friendly programming.

This work does not raise any ethical issues.

ACKNOWLEDGEMENT

We sincerely thank all the anonymous reviewers and the shepherd Robert Soulé for their valuable feedback and constructive suggestions on improving this paper. This work was supported in part by the National Key R&D Program of Zhejiang Province (2023R5202), the National Natural Science Foundation of China (NSFC) under No. 62302441, and Alibaba Cloud through Alibaba Innovative Research Program.

REFERENCES

[1] [n. d.]. BESS: Berkeley Extensible Software Switch. https://github.com/NetSys/bess.

[2] 1981. Internet Control Message Protocol. RFC 777. https://doi.org/10.17487/RFC0777

[3] 1983. The TCP Maximum Segment Size and Related Topics. RFC 879. https://doi.org/10.17487/RFC0879

[4] 2018. Virtual Switch Acceleration with OVS-TC and Agilio 40GbE SmartNICs. https://www.netronome.com/media/documents/WP_OVS-TC_40G.pdf.

[5] 2020. HCL and Intel's Open vSwitch acceleration solution. https://www.hcltech.com/sites/default/files/documents/inline-migration/hcls_ovs_acceleration_solution_brief-1-8-2020.pdf.

[6] 2020. Mellanox ASAP2 Accelerated Switching and Packet Processing. https://network.nvidia.com/files/doc-2020/sb-asap2.pdf.

[7] 2020. Using Stingray for OVS Offload. https://github.com/CCX-Stingray/Getting-Started.

[8] 2021. Logging IP traffic using VPC Flow Logs. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html.

[9] 2021. Virtual Switching Offload on FPGA-Based Alveo® SmartNICs. https://www.xilinx.com/publications/solution-briefs/partner/vvdn-ovs-solution-brief.pdf.

[10] 2022. A Detailed Explanation about Alibaba Cloud CIPU. https://www.alibabacloud.com/blog/599183?spm=a3c0i.23458820.2359477120.3.76806e9bESi3SD.

[11] 2022. Amazon Virtual Private Cloud Traffic Mirroring. https://docs.aws.amazon.com/vpc/latest/mirroring/what-is-traffic-mirroring.html.

[12] 2023. Broadcom TRUFLOW™. https://www.broadcom.com/solutions/data-center/cloud-scale-networking.

[13] 2023. Data Plane Development Kit. https://www.dpdk.org/.

[14] 2023. Intel® eASIC™ Devices. https://www.intel.com/content/www/us/en/products/details/asics/easics.html.

[15] 2023. Nginx Web Server. https://www.nginx.com/.

[16] 2023. Overview of Flowlog. https://www.alibabacloud.com/help/en/virtual-private-cloud/latest/flow-logs-overview.

[17] 2023. Redis. https://redis.io/.

[18] 2023. The Zeek Network Security Monitor. https://zeek.org/.

[19] 2023. Traffic mirroring overview. https://www.alibabacloud.com/help/en/virtual-private-cloud/latest/traffic-mirroring-overview.

[20] 2024. Falcon. https://cloud.google.com/blog/topics/systems/introducing-falcon-a-reliable-low-latency-hardware-transport.

[21] 2024. Iperf. https://iperf.fr/.

[22] 2024. Linux iptables. https://man7.org/linux/man-pages/man8/iptables.8.html.

[23] 2024. Linux Traffic Control. https://man7.org/linux/man-pages/man8/tc.8.html.

[24] 2024. Netfilter. https://www.netfilter.org/.

[25] 2024. Netperf. https://hewlettpackard.github.io/netperf/.

[26] 2024. Sockperf. https://github.com/Mellanox/sockperf.

[27] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-Scale Key-Value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK) *(SIGMETRICS '12)*. Association for Computing Machinery, New York, NY, USA, 53–64. https://doi.org/10.1145/2254756.2254766

[28] Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. 2020. hXDP: Efficient Software Packet Processing on FPGA NICs. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 973–990. https://www.usenix.org/conference/osdi20/presentation/brunella

[29] Qizhe Cai, Midhul Vuppalapati, Jaehyun Hwang, Christos Kozyrakis, and Rachit Agarwal. 2022. Towards μs Tail Latency and Terabit Ethernet: Disaggregating the Host Network Stack. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, 767–779. https://doi.org/10.1145/3544216.3544230

[30] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783710

[31] Sean Choi, Xiang Long, Muhammad Shahbaz, Skip Booth, Andy Keep, John Marshall, and Changhoon Kim. 2017. PVPP: A Programmable Vector Packet Processor. In *Proceedings of the Symposium on SDN Research* (Santa Clara, CA, USA) *(SOSR '17)*. Association for Computing Machinery, New York, NY, USA, 197–198. https://doi.org/10.1145/3050220.3060609

[32] Alex Conta. 1995. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6). RFC 1885. https://doi.org/10.17487/RFC1885

[33] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, 373–387. https://www.usenix.org/conference/nsdi18/presentation/dalton

[34] Chongrong Fang, Haoyu Liu, Mao Miao, Jie Ye, Lei Wang, Wansheng Zhang, Daxiang Kang, Biao Lyv, Peng Cheng, and Jiming Chen. 2020. VTrace: Automatic Diagnostic System for Persistent Packet Loss in Cloud-Scale Overlay Network. In *Proceedings of the ACM SIGCOMM 2020 Conference* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, 31–43. https://doi.org/10.1145/3387514.3405851

[35] Alireza Farshin, Tom Barbette, Amir Roozbeh, Gerald Q. Maguire Jr, and Dejan Kostic. 2021. PacketMill: toward per-Core 100-Gbps networking. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, Tim Sherwood, Emery D. Berger, and Christos Kozyrakis (Eds.). ACM, 1–17. https://doi.org/10.1145/3445814.3446724

[36] Daniel Firestone. 2017. VFP: A Virtual Switch Platform for Host SDN in the Public Cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 315–328. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/firestone

[37] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[38] Hamid Ghasemirahni, Tom Barbette, Georgios P. Katsikas, Alireza Farshin, Amir Roozbeh, Massimo Girondi, Marco Chiesa, Gerald Q. Maguire Jr, and Dejan Kostic. 2022. Packet Order Matters! Improving Application Performance by Deliberately Delaying Packets. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Amar Phanishayee and Vyas Sekar (Eds.). USENIX Association, 807–827. https://www.usenix.org/conference/nsdi22/presentation/ghasemirahni

[39] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 58–72. https://doi.org/10.1145/2934872.2934891

[40] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The P4->NetFPGA Workflow for Line-Rate Packet Processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, 1–9. https://doi.org/10.1145/3289602.3293924

[41] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. Association for Computing Machinery, 67–81. https://doi.org/10.1145/2872362.2872367

[42] Leonardo Linguaglossa, Dario Rossi, Salvatore Pontarelli, Dave Barach, Damjan Marjon, and Pierre Pfister. 2019. High-speed data plane and network functions virtualization by vectorizing packet processing. *Computer Networks* 149 (2019), 187–199. https://doi.org/10.1016/j.comnet.2018.11.033

[43] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. 2014. RFC 7348: Virtual EXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks.

[44] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve D. Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena E. Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, Tim Brecht and Carey Williamson (Eds.). ACM, 399–413. https://doi.org/10.1145/3341301.3359657

[45] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, and Hongqiang Harry Liu. 2022. From luna to solar: the evolutions of the compute-to-storage networks in Alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands)

(SIGCOMM '22). Association for Computing Machinery, New York, NY, USA, 753–766. https://doi.org/10.1145/3544216.3544238

[46] Michele Paolino, Nikolay Nikolaev, Jeremy Fanguede, and Daniel Raho. 2015. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. In *IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2015, San Francisco, CA, USA, November 18-21, 2015*. IEEE, 86–92. https://doi.org/10.1109/NFV-SDN.2015.7387411

[47] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open VSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) (NSDI'15)*. USENIX Association, 117–130.

[48] Boris Pismenny, Liran Liss, Adam Morrison, and Dan Tsafrir. 2022. The Benefits of General-Purpose on-NIC Memory. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. Association for Computing Machinery, 1130–1147. https://doi.org/10.1145/3503222.3507711

[49] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani Brunella, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, and Felipe Huici. 2019. FlowBlaze: Stateful Packet Processing in Hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 531–548. https://www.usenix.org/conference/nsdi19/presentation/pontarelli

[50] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Eric Peterson, Aaron Smith, Jason Thong, Phillip Yi Xiao, Doug Burger, Jim Larus, Gopi Prashanth Gopal, and Simon Pope. 2014. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*. IEEE Press, 13–24. https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/

[51] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: Congestion Signals Are Simple and Effective for Network Load Balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/3544216.3544226

[52] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, Boston, MA, 101–112. https://www.usenix.org/conference/atc12/technical-sessions/presentation/rizzo

[53] Luigi Rizzo and Giuseppe Lettieri. 2012. VALE, a switched ethernet for virtual machines. In *Conference on emerging Networking Experiments and Technologies, CoNEXT '12, Nice, France - December 10 - 13, 2012*, Chadi Barakat, Renata Teixeira, K. K. Ramakrishnan, and Patrick Thiran (Eds.). ACM, 61–72. https://doi.org/10.1145/2413176.2413185

[54] Luigi Rizzo, Giuseppe Lettieri, and Vincenzo Maffione. 2013. Speeding up packet I/O in virtual machines. In *Symposium on Architecture for Networking and Communications Systems, ANCS '13, San Jose, CA, USA, October 21-22, 2013*. IEEE Computer Society, 47–58. https://doi.org/10.1109/ANCS.2013.6665175

[55] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 123–137. https://doi.org/10.1145/2829988.2787472

[56] Rusty Russell. 2008. virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Oper. Syst. Rev.* 42, 5 (jul 2008), 95–103. https://doi.org/10.1145/1400097.1400108

[57] Hugo Sadok, Nirav Atre, Zhipeng Zhao, Daniel S. Berger, James C. Hoe, Aurojit Panda, Justine Sherry, and Ren Wang. 2023. Enso: A Streaming Interface for NIC-Application Communication. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, 1005–1025. https://www.usenix.org/conference/osdi23/presentation/sadok

[58] Alireza Sanaee, Farbod Shahinfar, Gianni Antichi, and Brent E. Stephens. 2022. Backdraft: a Lossless Virtual Switch that Prevents the Slow Receiver Problem. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Amar Phanishayee and Vyas Sekar (Eds.). USENIX Association, 1375–1392. https://www.usenix.org/conference/nsdi22/presentation/sanaee

[59] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostić, and Marco Chiesa. 2023. A High-Speed Stateful Packet Processing Approach for Tbps Programmable Switches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, 1237–1255. https://www.usenix.org/conference/nsdi23/presentation/scazzariello

[60] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (2020), 67–73. https://doi.org/10.1109/MM.2020.3016891

[61] William Tu, Yi-Hung Wei, Gianni Antichi, and Ben Pfaff. 2021. Revisiting the Open VSwitch Dataplane Ten Years Later. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, 245–257. https://doi.org/10.1145/3452296.3472914

[62] Gerry Wan, Fengchen Gong, Tom Barbette, and Zakir Durumeric. 2022. Retina: Analyzing 100GbE Traffic on Commodity Hardware. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, 530–544. https://doi.org/10.1145/3544216.3544227

[63] Chengkun Wei, Xing Li, Ye Yang, Xiaochong Jiang, Tianyu Xu, Bowen Yang, Taotao Wu, Chao Xu, Yilong Lv, Haifeng Gao, Zhentao Zhang, Zikang Chen, Zeke Wang, Zihui Zhang, Shunmin Zhu, and Wenzhi Chen. 2023. Achelous: Enabling Programmability, Elasticity, and Reliability in Hyperscale Cloud Networks. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, 769–782. https://doi.org/10.1145/3603269.3604859

[64] Yang Zhou, Ying Zhang, Minlan Yu, Guangyu Wang, Dexter Cao, Yu-Wei Eric Sung, and Starsky H. Y. Wong. 2022. Evolvable Network Telemetry at Facebook. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, Amar Phanishayee and Vyas Sekar (Eds.). USENIX Association, 961–975. https://www.usenix.org/conference/nsdi22/presentation/zhou