OSMOSIS: Enabling Multi-Tenancy in Datacenter SmartNICs

Mikhail Khalilov[1], Marcin Chrapek[1], Siyuan Shen[1], Alessandro Vezzu[1], Thomas Benz[2], Salvatore Di Girolamo[1], Timo Schneider[1], Daniele De Sensi[1,3], Luca Benini[2], and Torsten Hoefler[1]

[1]SPCL, D-INFK, ETH Zurich
[2]IIS, D-ITET, ETH Zurich
[3]Department of Computer Science, Sapienza University of Rome

Abstract

Multi-tenancy is essential for unleashing SmartNIC's potential in datacenters. Our systematic analysis in this work shows that existing on-path SmartNICs have resource multiplexing limitations. For example, existing solutions lack multi-tenancy capabilities such as performance isolation and QoS provisioning for compute and IO resources. Compared to standard NIC data paths with a well-defined set of offloaded functions, unpredictable execution times of SmartNIC kernels make conventional approaches for multi-tenancy and QoS insufficient. We fill this gap with OSMOSIS, a SmartNICs resource manager co-design. OSMOSIS extends existing OS mechanisms to enable dynamic hardware resource multiplexing on top of the on-path packet processing data plane. We implement OSMOSIS within an open-source RISC-V-based 400Gbit/s SmartNIC. Our performance results demonstrate that OSMOSIS fully supports multi-tenancy and enables broader adoption of SmartNICs in datacenters with low overhead.

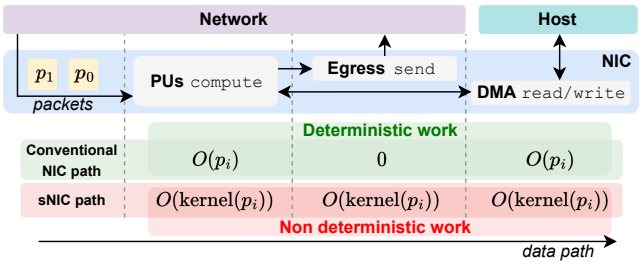

Figure 1: A predictable NIC data path versus the unpredictable sNIC kernel execution.

1 Introduction

Network data plane design has undergone two decades of exciting research, leading to the achievement of sub-microsecond packet processing host latency [8, 24, 26, 35, 38, 44–46, 60, 72, 76, 82]. SmartNICs (sNICs) have further improved processing times by enabling direct in-network packet processing, thereby reducing data movement [42]. sNICs started a trend in datacenter networking acceleration [47, 92] similar to the GPU trend in high-performance computing [94].

sNICs enable running *kernels* on programmable, energy-efficient cores tailored for packet processing and integrated within the host network interface card (NIC) System-on-Chip (SoC). These cores are attached directly (i.e., *on-path*) to the datacenter Ethernet or InfiniBand link [5, 54]. Such a design reduces the latency of some applications since the sNIC can process the packets in the network [58] and reply directly without moving the packets to/from the host OS networking stack [1, 31]. This enables the offload and acceleration of several workloads such as distributed learning gradients aggregation [89, 94], disaggregation and storage [28, 30, 49, 62, 63], Key-Value Stores (KVS) [75, 88, 100], Remote Procedure Calls (RPCs) [14, 53, 57, 79, 98], network protocols and telemetry [14, 16, 22, 39, 64, 85, 98, 99].

Network resources in a datacenter are multiplexed between tenants through a virtualization layer [12, 17, 51, 66, 102]. However, processing user code by sNICs brings a set of considerable resource management issues. As Figure 1 shows, NICs have three resources that must be multiplexed: compute, Direct Memory Access (DMA) bandwidth, and egress bandwidth. The traditional NIC data path only forwards packets to host memory and executes simple operations with a *predictable* and *bounded* complexity. Typically, the number of incoming bytes equals the number of outgoing bytes, and NICs do not run any elaborate processing on them. In contrast, sNICs can execute *unpredictably* complex stateful offloads [74]. For example, heavily used in machine learning [9] Allreduce operates on the payload and is compute-bound, while storage offloading predominantly accesses host memory and is DMA/IO bound. sNICs need to operate on *uncoordinated*, *non-deterministic*, and *concurrent* data streams while meeting Service Level Objective (SLO) policies set by the administrator.

Achieving a fair resource multiplexing for sNICs is chal-

lenging. sNICs combine characteristics of an accelerator, such as a GPU, and a traditional NIC. While this provides the aforementioned benefits, the resource management of neither is directly applicable due to the unique sNIC requirements (Section 3). Conventional RDMA NICs (rNICs) have bounded and predictable workloads (e.g., atomics, scatter-gather RDMA reads/writes) and often use link bandwidth allocation as a *"just enough"* mechanism for resource isolation and Quality-of-Service (QoS) between tenants. Although rNICs exhibit bounded and foreseeable behavior, achieving fairness is a challenging endeavor [96] even within their simpler than sNIC context. In contrast, accelerators fall entirely under the governance of the host OS, which oversees all active kernels [48,50]. These accelerators neither generate nor receive events beyond instructions from accelerated applications, setting them apart from sNICs capable of executing arbitrary kernels independently of the host's involvement.

Furthermore, for sNICs to sustain the sub-*nano*second packet arrival intervals at fully utilized 400Gbit/s link (Section 3, [27]), resource multiplexing must be conducted fast. On-path sNICs have much stricter compute and buffering constraints than traditional NICs and accelerators due to the packet rate and the three multiplexed resources (compute, DMA, and egress). This issue is even more critical as network rates constantly increase and are expected to exceed Terabit per second by 2025 [15,23,33,91].

A common approach to effectively manage processing at high packet arrival rates involves implementing resource management in hardware [2,4,27]. This is usually accomplished through scheduling policies such as Weighted Round Robin (WRR), which divide link bandwidth among tenants [19,20,96]. However, because sNICs have varying application kernel requirements, incorporating WRR for compute resource allocation can lead to unfairness. For example, as we show in Section 3, if one application (e.g., Allreduce) is compute-bound and takes twice as much compute time as a non-compute-bound application (e.g., KVS), the former will be able to process twice as many bytes. Other recently proposed methods for compute isolation in sNICs are not optimal for all scenarios as they are either non-work conserving [29] or rely on the host CPU as a fallback path [57].

We tackle these issues by introducing OSMOSIS (Operating System Support for Streaming In-Network Processing) (Section 5). OSMOSIS is a lightweight sNIC management layer that supports performance-critical data-plane management in hardware and non-critical management tasks in a flexible software runtime. OSMOSIS is a fair, work-conserving sNIC resource manager that requires minimal hardware footprint and employs expressive yet simple Service Level Objective (SLO) semantics. In OSMOSIS, the sNIC is exposed to a tenant as Single-Root Input/Output Virtualization (SR-IOV) Virtual Function (VF). This allows the administrator to allocate proportionally more *compute processing units*, *egress bandwidth*, and *DMA bandwidth* to VFs associated with high-priority tenants.

We implement (Section 6) and evaluate (Section 7) OSMOSIS on top of one of the available open-source on-path sNIC architectures, PsPIN [18,32]. PsPIN is based on energy-efficient silicon-proven RISC-V cores. In our setup, PsPIN is the hardware backbone for packet processing using kernels written in C. Our performance evaluation focuses on typical datacenter workloads such as storage IO and in-network Allreduce, and shows that OSMOSIS provides comprehensive support for multi-tenancy without sacrificing performance.

In summary, we make the following contributions.

1. *sNIC multi-tenancy:* We show typical multi-tenancy sNIC problems and define a set of requirements for high-performance sNICs. These requirements serve as a guideline for developing sNICs that can meet the needs of diverse workloads and tenant environments (Section 3).

2. *OSMOSIS:*

   We introduce OSMOSIS, a lightweight sNIC resource manager based on fair and work-conserving scheduling policies. OSMOSIS is a minimal hardware footprint solution to the problem of fair and efficient resource sharing in multi-tenant sNICs with diverse application needs (Section 5).

3. *Evaluation:* We implement OSMOSIS in an open-source on-path 400Gbit/s sNIC by extending it with schedulers and a control path prototype (Section 6). We use this implementation to verify and evaluate OSMOSIS. We demonstrate how it solves the defined sNIC problems and handles multi-tenant applications fairly with varying resource requirements while minimizing tail latency (Section 7).

2 Background and Related Work

From the system's perspective, we abstract out the sNIC as a packet processing accelerator between the network fabric and the host CPU, GPU, or FPGA.

Existing sNICs can be classified broadly into two categories: *off-path* and *on-path* [57].

Off-path sNICs add an entire CPU complex to the network card, often running a full operating system (e.g., Linux). This design enables a management plane based on receive side scaling (RSS) to be conveniently implemented [8, 61, 76]. However, they often suffer from lower performance in terms of latency, bandwidth, and packet processing rates due to their system design, which closely resembles the CPU-centered host architecture (e.g., Broadcom Stingray and Nvidia Bluefield data processing units (DPUs) are both feature ARM SoCs with PCIe and DRAM).

On-path sNICs share packet input buffers with *processing units* (PUs) tailored for packet processing (e.g., LiquidIO [59],

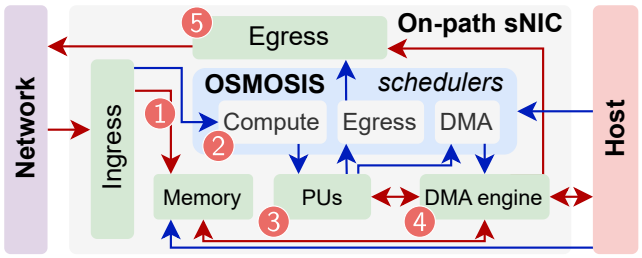

Figure 2: Schematic overview of on-path sNIC architectures. Red arrows indicate the data path and blue arrows correspond to the control/management path.

Netronome [68], PsPIN [18], Data Path Accelerator (DPA) introduced in Bluefield 3 DPU [69, 70]). On-path sNICs typically provide programming API for writing *kernels* that process traffic on PUs, either on per-packet (PsPIN [18]) or per-message granularity (Bluefield-3 FlexIO API [69]). PUs typically feature three layers of the memory hierarchy, e.g., L1 single-cycle access scratchpad, L2 memory with access latency of 15-50 cycles, and host side memory (either off-path SoC or host CPU memory). L1 and L2 memories could be organized as multi-level caches (e.g., LiquidIO) or be explicitly managed by the user (e.g., PsPIN).

OSMOSIS provides a solution to a fair resource multiplexing for sNICs in a multi-tenant context and is not specific to any system. However, to showcase the identified issues and verify and evaluate the overhead of OSMOSIS, we selected one of the possible synthesizable open-source on-path sNIC implementations available in the literature, namely, PsPIN. We decided to use an on-path sNIC as our experiments (Table 1) show that only such sNICs can sustain packet processing at emerging line rates. PsPIN is open-source, based on energy-efficient silicon-proven RISC-V cores, and allows users to write packet processing kernels in C and explicitly manage sNIC memories [18]. OSMOSIS could have been equivalently implemented in any other sNIC framework [59, 68, 69].

2.1 Challenges of Resource Isolation

We generalize on-path sNIC architecture in Figure 2. Packets arrive at the sNIC inbound engine (1) and are initially stored at the L2 packet buffer organized as a set of per-application first-in-first-out (FIFO) queues. Next (2), packets are scheduled for processing on available PUs where kernel execution is initiated (3). Kernels execute using three resources, PUs, DMA, and Egress bandwidth. Each application uses these resources differently (e.g., compute- or IO-bound) depending on its needs. In general, these resources can be used as follows:

(3) PUs: computing (e.g., hashing the packet header or summing values in an Allreduce reduction);

(4) DMA engine: transferring data to read/write in sNIC memory (e.g., KVS cache in sNIC L2 memory) or host memory (e.g., KVS cold storage);

(5) Egress engine: sending packet replies (e.g., reply to a read request with a value from the KVS cache).

Metrics to measure the quality of resource multiplexing by datacenter tenants, known as Service-Level Objectives (SLOs), are typically tied to the conventional NIC path displayed in Figure 2 by considering tail latency [17] and throughput [67, 84]. However, these SLOs do not consider the sNIC data path with its unique resource multiplexing discussed in Section 3, such PU time, tail latency of DMA over host interconnect, and buffer space. Existing proposals have only partially addressed this issue by introducing performance isolation mechanisms, such as multi-level packet scheduling [27, 57, 87] and static resource allocation [29] of shared resources (see Section 4). Yet, due to the kernels' dynamic and unpredictable nature, static assignments do not solve the problem. OSMOSIS fills this gap by *providing bounded guarantees for the sNIC resource availability to tenants using dynamic resource multiplexing*.

3 Multi-Tenant sNICs

Diverse application requirements create distinct resource multiplexing bottlenecks. Our quantitative analysis highlights these issues in multi-tenant setups of existing sNIC stacks [18, 69], yielding sNIC requirements. These insights directly led to the microarchitectural and software choices for OSMOSIS. We use a 400 Gbit/s link for all experiments (more details on the setup in Section 7).

Per-packet time budget (PPB): While studies of datacenter traffic show that only a fraction of the established connections actively exchange data at any given time [10, 81, 97], they can still saturate the link bandwidth. To analyze the implications of this for sNICs we define per-packet time budget (PPB) using PU count N, packet size P, and link bandwidth B as $PPB(N,P,B) = N \times (P/B)$. In this case, we model the sNIC as a $M/M/m$ queue where PPB defines the condition which needs to be satisfied for the queue to be stable [13][1]. To be more specific, PPB represents how long the sNIC can process a packet until the next one arrives, assuming a fully utilized link. If PPB is exceeded, the per-application ingress queue will eventually fill up during transient traffic bursts leading to packet drops or falling back to link flow control (e.g., PFC [103]) and a possible violation of per-VF SLO policy.

Figure 3 compares service times of IO– and compute-bound workloads with theoretical PPB assuming that tenant workloads fit one packet and that the sNIC has only one tenant.

[1] $1/\lambda = P/B$, $m = N$, to achieve $\rho < 1$, $1/\mu > N \cdot P/B$, where PPB = $1/\mu$.

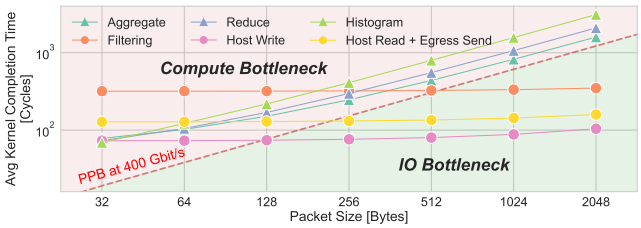

Figure 3: sNIC core (PU) processing time needed to serve 1 packet for common sNIC kernels. Workloads with triangle markers are compute-bound, and circular markers are IO-bound. All workloads with $\leq$ 64B packet size (including 28 bytes IPv4/UDP-header) exceed PPB showing congestion at PUs when link bandwidth is fully utilized. Note that our setup supports Ethernet payload sizes below 64B to accommodate custom interconnects [41].

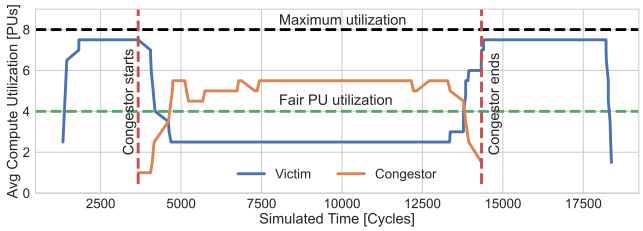

Figure 4: *Congestor* and *Victim* tenants' flows with equal priorities are mapped to two different SR-IOV VFs with equal shares of Ingress bandwidth. With the round-robin scheduling of per-flow queues, the *Congestor* tenant with 2$\times$ higher compute cost per packet occupies a proportionally larger number of cores than the *Victim* tenant.

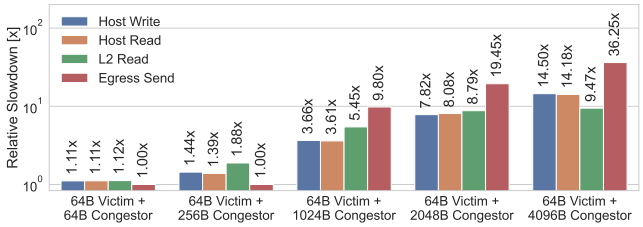

Figure 5: Slow-down of various IO operations (e.g., DMA and sending packets to Egress) initiated by the tenant's kernel results in HoL-blocking small requests due to underlying IO path contention.

We observe that all workloads with packet size $\leq$ 64 Bytes fail to fit in PPB. Compute-bound workloads (i.e., Aggregate, Reduce, Histogram) whose execution time scales linearly with packet payload length exceed the PPB for all packet sizes bottlenecking the PUs. Notably, IO-bound kernels above 256 Bytes (i.e., DMA writes/reads, Egress packet sends) fit PPB as they avoid PU congestion but are bottlenecked by the link bandwidth. However, as we will demonstrate, *IO-bound workloads are sensitive to DMA transfer contention on the host interconnect*.

PU contention: While a single tenant can cause pressure on the ingress queue and contention of PUs, multiple tenants can lead to unfairness. For example, consider two compute-bound tenants with different requirements. One of them, the *Congestor*, has twice as large compute cost per packet as the other, the *Victim*, leading to twice as many cycles on PU to finish the kernel. During the burst, *Congestor* and *Victim* push packets at the corresponding per-application (per-VF) queues at the same ingress rate. As Figure 4 shows, using the conventional round robin (RR) scheduling of per-application queues across 8 sNIC PUs, the *Congestor* uses 2$\times$ the PUs used by the *Victim*.

R1 *sNIC manager should fairly allocate compute components (e.g., PUs, cryptographic accelerators) while serving tenants with different compute costs per packet.*

Egress and DMA engines contention: Similarly, as the compute-bound kernels cause contention on PUs, IO-bound kernels can lead to contention on the appropriate DMA or egress engines. IO-bound kernels running on different PUs can simultaneously initiate IO requests through the same sNIC engines, e.g., DMA requests from a KVS application. In case the underlying interconnect (e.g., PCIe or AXI [78]) is blocking and lacks the support of QoS provisioning, *the issue of multiple concurrent requests may result in Head-of-Line (HoL) blocking* [1].

For example, consider two IO-bound tenants with different IO requirements. The *Victim* has constant 64B packets, while the *Congestor* increases its packet size from 64B to 4096B. As Figure 5 shows, the contention on the IO engine leads to an order of magnitude higher latency of the *Victim*'s messages without considerably affecting the *Congestor*'s flow. This unfairly increases the latency of one of the tenants by 4-15$\times$.

R2 *sNIC manager should fairly allocate DMA and egress bandwidth (e.g., using AXI and PCIe) between running kernels and be resilient to HoL-blocking.*

Memory management: Applications have diverse memory runtime needs, with dynamic memory allocation causing an unknown *a priori* memory consumption. In extreme cases, a tenant could monopolize all sNIC memory, e.g., L1 packet buffers, resulting in HoL-blocking for others. Introducing virtual memory (paging) semantics could lead to substantial memory access overheads, as each page fault significantly amplifies memory access latency [37].

R3 *sNIC manager should fairly allocate memory using lightweight allocation strategies defined in the control plane.*

Scheduling overhead: Existing *software* packet processing data paths [8, 24, 76] were designed for off-path sNICs or con-

PU	Frequency	ISA	Linux	Caladan	RTOS
Host Ryzen 7 5700	3.8GHz	x86	28576	211	–
BF-2 DPU A72	2.5GHz	ARMv8	13250	192	–
PULP cores [6] (used in PsPIN)	1GHz	RISC-V	–	–	121

Table 1: Average latency of context switching between 2 processes. Measurements shown in PU cycles scaled to 1 GHz (i.e., 1 ns/cycle).

ventional host processing. As recent studies show [44] *effectiveness* of kernel execution scheduling in terms of achieved maximum utilization while running on off-path sNICs supported by OS's like Linux is driven by the latency of context switching [26, 44]. PU cycles are wasted during context switching to transition between the kernel states. We benchmark context-switching of Linux running on host and off-path sNIC (Bluefield-2 ARM SoC). We compare these to the state-of-the-art Caladan scheduler we ported to the ARM ISA [26]. For reference, we also show the context switching latency of PULP cores as implemented in PsPIN used to evaluate OSMOSIS. Notably, we observe that the context switching latencies we report in Table 1 are higher or of the same order of magnitude as the PPB from the analysis presented in Figure 3.

R4 *Data path performance should not be impacted by overheads stemming from software scheduling policies, providing low-latency scheduling of kernel execution.*

Control path priority: If a tenant on the sNIC exceeds compute or time budgets, an immediate response is needed from the host's control plane for *control traffic*. However, communication between sNIC and host uses system interconnect (e.g., PCIe), typically adding an overhead of 0.5 – 2 usec per read-/write request. Congestion in the interconnect (Figure 5, [1]) can lead to HoL-blocking of control traffic and unpredictable packet processing. Moving request execution to the host (e.g., iPipe [57]) to resolve this introduces added latency overheads for packet processing.

R5 *sNIC accelerated packet processing should prioritize control-path traffic and not rely on latency-introducing host CPU as a fallback path.*

QoS API: NIC capabilities are exposed to tenants through a virtualization layer (OS hypervisor) that provides an illusion of full resource ownership. SR-IOV is a conventional way to implement NIC virtualization. In SR-IOV, each NIC physical function (PF) (such as TX and RX capabilities) is multiplexed between several virtual functions (VFs). Each VF is exposed to the tenant through an OS hypervisor as a stand-alone PCIe NIC. To our knowledge, existing production rNICs and sNICs support only Ingress and Egress bandwidth allocation on the coarse basis of VFs and not compute or DMA resources.

R6 *sNIC management plane should support conventional QoS provisioning mechanisms for all types of resources.*

4 Existing Solutions are Insufficient

We summarize recent resource management research milestones for NIC resource management in Table 2.

Justitia [102] and PicNIC [51] are rNIC virtualization layers lacking on-NIC compute management. They function as software controllers between the NIC and host application, handling RDMA read/write operations atop the RDMA API. Lynx [90] focuses on sNIC GPU data movement offloading but similarly manages traffic at a per-message granularity and lacks detailed multi-tenancy issues analysis.

Floem [73], FairNIC [29], and iPipe [57] specifically target on-path SmartNICs programmability. FairNIC aims for multi-tenant use cases by statically allocating compute and IO bandwidth to flows. Such an approach can potentially cause under-utilization or unfairness [44, 76, 82]. Notably, all three solutions lack flow priorities implementation. Per-flow priority is found in PANIC [56] and Menshen [93], yet both solutions are specialized for Reconfigurable Match Tables (RMT) pipeline architecture with P4 programming model.

In contrast, OSMOSIS is a general-purpose resource manager for on-path offloading, applicable to C/C++ applications, and offers dynamic work-conservative load balancing of sNIC PUs with adjustable tenant/flow priorities.

	Fairness			Efficiency		Deployment
	R1	R2	R3	R4	R5	R6
Floem [73]	▭	▭	◧	▭	▭	▭
Lynx [90]	▭	▭	▭	▭	■	◧
PicNIC [51]	▭	■	◧	▭	▭	◧
PANIC [56]	◧	◧	▭	■	▭	▭
Justitia [102]	▭	■	◧	▭	▭	■
iPipe [57]	◧	◧	■	■	▭	▭
Menshen [93]	◧	◧	■	■	◧	▭
FairNIC [29]	◧	◧	◧	■	■	■
OSMOSIS	■	■	■	■	■	■

Table 2: State-of-the-art proposals for on-path NIC resources management and supported design requirements.
■: supported, ◧: partially supported, ▭: not discussed/-supported.

5 OSMOSIS

We present OSMOSIS in Figure 6. We start by outlining how OSMOSIS handles sNIC resources and fulfills multi-tenancy requirements. This involves a two-part design: a host-based flexible software control plane for management and a

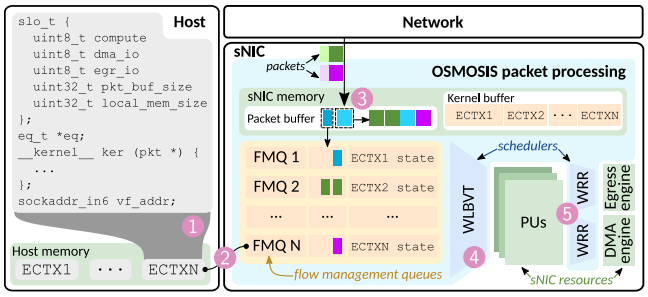

Figure 6: Abstract model of OSMOSIS-enabled sNIC. Packets are mapped by Matching Engine to FMQs and dispatched for execution by scheduler.

hardware data plane for SLO policy enforcement. Within this section, each part is explained in depth.

	PUs	DMA	Egress	Memory
Scheduler	WLBVT	WRR	WRR	Static
SLO knob	Priority Kernel cycle limit	Priority	Priority	Allocation size
Fulfilled requirements	R1 R4 R6	R2 R4 R5 R6	R2 R4 R6	R3 R4 R6

Table 3: OSMOSIS resource management principles with all six fulfilled multi-tenancy requirements.

5.1 High-level Overview

1 Flow execution context creation: To utilize sNIC packet processing, tenants create a flow *execution context* (ECTX). ECTX encapsulates the flow processing state, such as the SLO policy and the packet processing *kernel*, a piece of code compiled for the target PU architecture and describing the actions for each packet destined for the flow.

2 ECTX initialization: After the tenant provides the basic elements of an ECTX, OSMOSIS instantiates it. It allocates a virtualized sNIC interface through the host OS hypervisor and associates it with a tenant IP address and SLO policy. It also sets up the IOMMU to allow kernel access to specific host pages, *statically* allocates on sNIC memory and loads the kernel binary into sNIC memory.

3 Matching packets to flow management queue: The sNIC matching engine filters packets that require sNIC processing. All incoming packets are matched against the three-tuple (in case of UDP) or five-tuple (in case of TCP) of active sNIC ECTXs. Once matched, *packet descriptors* (e.g., pointer to packets in sNIC memory) are stored at one of the *flow management queues* (FMQs). FMQs store all information regarding an active flow ECTX on the sNIC hardware. FMQs are organized as FIFO queues of packet descriptors with an additional memory state to store running execution information (e.g., BVT metric).

4 PU scheduling: Once a PU becomes available, OSMOSIS schedules the packet at the head of one of the FMQs. To achieve fair PU allocation, OSMOSIS implements a centralized, non-preemptive scheduler inspired by the Borrowed Virtual Time (BVT) policy [21, 44]. BVT aims to allow each tenant to obtain the same amount of access time to the scheduled resource by keeping track of their past usage. OSMOSIS FMQ scheduler *allocates sNIC PUs to FMQs with the smallest priority-adjusted past PU usage measured in cycles* while maintaining the SLO policy specified by the sNIC administrator, such as the upper per-FMQ PU limit.

5 Kernel execution and IO management: Upon loading the packet into local PU memory, the PU can process it using the relevant kernel. As seen in Section 3, parallel kernel executions on different PUs can lead to head-of-line blocking (HoL-blocking) and uncertain tail latency for DMA to sNIC/host memory and egress data transfers. For example, kernels can pipeline large storage reads by overlapping asynchronous DMA reads of packet-sized payloads with egress packet sending. OSMOSIS mitigates this by fairly arbitrating IO paths, breaking sizable DMA requests into smaller transactions, and scheduling them with a near-perfect fairness-weighted round-robin (WRR) policy. FMQs supply DMA and egress engines with tenant IO priorities for initiated IO requests. This ensures each tenant obtains a priority-based fair bandwidth chunk.

5.2 Flexible software control plane

OSMOSIS offers a host OS API for sNIC packet processing management, encompassing ECTX creation and offloading specific flow handling to the sNIC. Tenant-initiated offloading involves the creation of a flow ECTX. ECTX facilitates tenant control using the following components.

SLO policy: The SLO policy sets compute, DMA, and egress priorities, per-kernel cycle budget, packet buffer size, and on-sNIC memory. OSMOSIS offers transparent SLO management via SLO knobs indicated in Table 3. By default, all tenants' FMQs share equal priority. To achieve perfect fairness in such a scenario, all flows should get the same portion of PUs and IO bandwidth at any time. Increasing the priority of the ECTX leads to *proportionally* more resources (PUs, bandwidth) allocated to the ECTX. A per-kernel cycle limit curbs excessive PU usage, adjustable for total or individual kernel execution times. We assess priority's impact on resource fairness in Section 7.

Kernel binary: kernel binary cross-compiled by the tenant is loaded into sNIC memory by the control plane and is later executed on the flow packets. The kernel binary can compute and schedule DMA and egress requests according to the tenant requirements.

A virtualized sNIC device: A virtualized device is allocated

for the tenant by OSMOSIS, e.g., SR-IOV VF. OSMOSIS associates an IP address with the virtualized device and uses it later for matching. The virtualized device is connected internally with a single FMQ.

A matching rule: The matching rule matches packets from the sNIC inbound stream to the ECTX and manages their processing within the same FMQ. A matching rule allows the tenants to open multiple ports on the same virtualized device. The matching engine can match packets based on their UDP/TCP header contents. For example, it can match the IP address and the destination port of the application.

sNIC memory segments: The sNIC memory segments are allocated statically to each kernel depending on the requested memory size. The kernels can store the application state in sNIC local memory, e.g., KVS-cache or packet filter table. The minimum allocation a valid ECTX has is the size of the kernel binary loaded into the sNIC memory by the control plane. An error is returned if the tenant uses too much memory or the kernel binary is larger than the SLO policy limits.

Host memory pages: The ECTX specifies which host pages can be accessed from the specific kernel via DMA. The DMA engine on the sNIC interfaces the host memory with an IOMMU, translating host virtual addresses to physical addresses. The IOMMU also checks whether the sNIC is accessing an allowed memory region. The control plane initializes the IOMMU with appropriate page tables during execution context creation.

Event queue (EQ): An event queue allows the user application to track events like kernel execution errors. When an error occurs (e.g., illegal memory access or exceeding execution time), OSMOSIS informs the host via an event in the kernel's ECTX EQ. A host OSMOSIS API call from the application checks this queue for error messages. EQ can be realized as contiguous sNIC memory mapped to the host virtual address space, akin to RDMA Verbs API EQ [41]. EQ control path traffic shares the sNIC DMA data path (e.g., PCIe or CXL) with regular kernel execution (e.g., DMA initiated within the kernel) but gets the highest IO priority due to tenants' immediate action needs.

5.3 Hardware data plane

OSMOSIS provides low management overhead with a minimal hardware footprint. We present two key mechanisms that help us to achieve this goal: a hardware flow abstraction (FMQs) and scheduling algorithms suitable for hardware implementation (WLBVT and DWRR).

Flow management queues (FMQs) generalize a packet flow similarly to how a hardware thread generalizes a process. FMQs store matched packet descriptors in a FIFO queue and monitor the flow processing performance. The scheduler then uses these measures to allocate compute resources fairly and enforce per-flow priorities. Processing the FIFO queue triggers kernel executions on sNIC PUs, resembling program instruction execution flow in traditional OS processes.

FMQs also store part of the ECTX state, such as the matching rule, pointers to the kernel binary, and the SLO policy definition. The host-side control plane manages and initializes FMQs that appear as MMIO registers in SR-IOV VF address space. FMQs are highly extensible. For example, the OSMOSIS priority model is compatible with datacenter Ethernet [40]. In case of congestion on the FMQ FIFO queue, the packets can be marked with the appropriate Ethernet ECN congestion flag or can supply the per-FMQ telemetry information [2, 3, 25, 41, 55, 103].

FMQ Scheduler allocates PUs across flows with different compute, DMA, and egress costs-per-packet that are not known *a priori*. Thus, to achieve fair compute utilization, the FMQ arbitration policy needs to be *invariant to the cost-per-byte of the packet* (see Figure 4). OSMOSIS implements a hardware scheduler as simple and scalable as the deficit-weighted round-robin (DWRR) but with a minimal additional area footprint (see Section 6).

OSMOSIS utilizes a greedy *Weight Limited Borrowed Virtual Time* (WLBVT) policy, a hybrid of the Weighted Fair Queuing (WFQ) model of FMQ weights and Borrowed Virtual Time (BVT) scheduler. We adopt the BVT algorithm to suit sNIC hardware implementation constraints [21, 44] and present our scheduler in pseudo-code Listing 1. Intuitively, our scheduler aims to allocate each tenant the same amount of PU processing time normalized by priority while ensuring that each tenant is served fairly during PU contention.

```
def pu_limit(ActiveFMQs, fmq):
  prio_sum = 0
  for fmq in FMQs:
    if not fmq.empty:
      prio_sum += fmq.prio
  return ceil(len(FMQs) * fmq.prio / prio_sum)

def update_tput(FMQs): #called at each clock cycle
  for fmq in FMQs:
    fmq.total_pu_occup += fmq.cur_pu_occup
    if not fmq.empty or fmq.cur_pu_occup > 0:
      fmq.bvt += 1 # update only in active state
    fmq.tput = fmq.total_pu_occup / fmq.bvt

def get_fmq_idx(): #called once PU core is free
  min_tput = MAX_INT
  for fmq in ActiveFMQs:
    if fmq.pu_occup < pu_limit(activeFMQs, fmq):
      if fmq.tput / fmp.prio < min_tput:
        min_tput = fmq.tput / fmq.prio
        fmq_idx = fmq.idx
  return fmq_idx
```

Listing 1: WLBVT FMQ scheduler procedural pseudocode.

An FMQ is in an active state if it contains packet descriptors in the FIFO queue or if its packets are currently being processed on any PU. Flow throughput is updated (`update_tput`) at each sNIC clock cycle only if the corre-

sponding FMQ is active. The scheduler (`get_fmq_idx`) returns the index of the non-empty FMQ that fits the upper limit of weighted PU occupation (`pu_limit` called in line 21) and has the lowest current throughput normalized by FMQ priority (lines 22, 23).

The weighted PU occupation's upper limit guarantees fair QoS for tenants based on their priority. `pu_limit` is calculated with a *ceil* function to ensure fairness in case of more active FMQs than PUs or non-integer division. The lowest priority normalized throughput equalizes access to oversubscribed PUs over time, favoring lower resource usage users. Our approach can also accommodate total virtual time per tenant (i.e., line 21), which could be useful for billing purposes, thus expanding policy flexibility.

Kernel execution is a short-lived event as each execution only processes one packet. In OSMOSIS, we run kernels to completion [8, 76]. We avoid context-switching for several reasons. As shown in Table 1, context switching can introduce significant overhead. It also increases the complexity of the hardware data path and requires additional states per each active kernel.

If a kernel exceeds a set time limit (e.g., per-FMQ watchdog timer), it's terminated with a hardware interrupt, and the host application receives notification via the corresponding EQ. We believe that run-to-completion semantics underpins the sNIC programming model that, together with OSMOSIS fair priority adjusted schedulers, ensures predictable packet processing tail latency and also excludes compute-intensive tasks better suited for GPUs or FPGAs [8, 76].

5.4 Discussion

Encrypted traffic: The sNIC handles data movement and may also require accessing the packet contents. Hence, it should be able to decrypt packets (e.g., QUIC [99]). sNICs can support either per-PU cryptographic accelerators (e.g., Intel AES-NI [34]) or a shared accelerator for efficiency (e.g., like in Marvell LiquidIO [59]) exposed via ISA extensions. In the latter case, the cryptographic accelerator arbitration resembles PUs, making WLBVT scheduling suitable for access management.

IO security: Host memory is protected against unauthorized DMA transfers using an IOMMU setup by OSMOSIS when the host creates the flow context. Similarly, local sNIC memory accesses need to be protected. This can be achieved, for example, by a *Physical Memory Protection* unit (PMP) [95] as shown in Section 6.1.

Transport protocols: While this work does not focus on sNIC transport protocols, OSMOSIS, by design, is compatible with conventional congestion signaling (e.g., ECN) and lossless flow control mechanisms (e.g., Ethernet DCB). It can also be deployed with DCQCN [103] and DCTCP [3]. From the transport protocol perspective, the packet queueing delay within the FMQs and the corresponding execution of the packet kernel is just another source of latency. For example, the FMQ abstraction deployed with Ethernet can support RED/ECN marking [25, 41]. Another mechanism that FMQ can easily support is supplying the P4 INT-MD telemetry information [2] to enable the HPCC protocol [55].

6 Implementation

We implement OSMOSIS atop PsPIN [18, 32], an open-source on-path sNIC. We adopt PsPIN as a backend for performance-critical operations within OSMOSIS by extending its host-side API to support multiple ECTXs and specify tenant SLOs using 335 lines of code (LOCs) in C. We integrated functional blocks of OSMOSIS (i.e., matching engine, WLBVT scheduler, and DMA request fragmentation) written in 1216 LOCs of C++ with cycle-accurate simulation PsPIN SystemVerilog backend. In addition, we also implemented these components as synthesizable SystemVerilog IP blocks for hardware cost estimations. These blocks can serve as a future prototype for ASIC or FPGA-based implementation of OSMOSIS.

6.1 Implementing OSMOSIS on top of PsPIN

Packet processing units: OSMOSIS PsPIN architecture is based on scalable silicon-proven RISC-V PULP SoC [18, 52, 80]. The PUs are RI5CY 32-bit cores organized in clusters. Each PsPIN cluster contains 8 PUs clocked at 1GHz and coupled with a 1 cycle, multi-banked local scratchpad memory (referred to as *L1*). For our experiments, we use the default configuration of the PsPIN PU cluster with 1 MiB L1 data, and 4 KiB L1 instruction caches. Clusters share a global 4 MiB L2 packet buffer and a 4 MiB L2 kernel buffer, which can be used for local data storage.

Portable programming API: OSMOSIS utilizes PsPIN infrastructure to offload the processing packets to the PUs. The user writes a C kernel cross-compiled on the host for the RISC-V ISA architecture. The kernels are then loaded and executed on the flow packets according to the sPIN API [32].

Kernel IO: The PsPIN API enables blocking and non-blocking IO calls within kernel code. Each PsPIN cluster has a 512-bit AXI DMA interconnect connecting cluster scratchpad memories to the sNIC L2 kernel buffer, host DMA engine buffer, and sNIC egress engine buffer. This setup enables `read` and `write` transfers between these buffers, with PUs accessing other cluster memories and shared L2 kernel memory in 10 to 30 cycles. This design also transparently supports sNIC egress packet `send`: a DMA `write` from kernel scratchpad memory to the NIC egress engine buffer. PU core L1 scratchpad interfaces an Ethernet egress pipeline over the AXI protocol. PsPIN IO-calls configure a DMA command

with addresses, length, and a completion handle pointer. The cluster command FIFO queues outstanding IO commands, and a WRR policy arbitrates per-cluster queues for DMA engine access.

Memory management: Our implementation allows to specify the size of the L2 and L1 memories allocatable to tenants. We implement memory isolation using Physical Memory Protection (PMP) PsPIN PU unit. When the kernel accesses L1 and L2 memories, the virtual memory addresses are translated to physical addresses with relocation registers. The PMP then checks that the addresses are within the valid segment range. Like the relocation registers, the PMP unit does not increase the memory access latency [18].

6.2 OSMOSIS Schedulers

FMQ scheduling implementation: FMQ encompasses a FIFO queue, ECTX (detailed in Section 5), and scheduling state. The FIFO queue holds packet descriptors, each containing a 32-bit pointer to the packet. The scheduling state includes a BVT counter tracking tenant resource use and a priority. We implemented the counter as a 64-bit register to avoid overflow[2]. A 16-bit register stores the FMQ priority. Our SystemVerilog WLBVT implementation with 128 FMQs synthesizes at 1 GHz, making a scheduling decision in five cycles. Most latency stems from the weight-limiting requiring integer division which is challenging for fast hardware implementation. We hide this latency using pipelining, overlapping FMQ arbitration with packet DMA from L2 packet buffer to the cluster scratchpad (at least 13 cycles for a 64-byte packet).

Enhanced DMA engine: To prevent HoL-blocking, OSMOSIS applies transfer fragmentation on both the host-interfacing DMA engine and the egress engine. We implement two modes of fragmentation: a *software* fragmentation implemented within the kernel call for a DMA transfer and a *hardware* fragmentation within the DMA engine. The software approach wraps `pspin_dma_read/write` and `pspin_send_packet` with a function, dividing larger requests into smaller chunks. We issue multiple non-blocking DMA requests of smaller sizes while internally maintaining the state for each transfer. While this optimization mitigates HoL-blocking (as shown in Section 7), it also hinders the throughput of large DMA requests. To minimize this, we expand the functional model of AXI to enable hardware DMA fragmentation offloading. This involves managing the state for multiple outstanding AXI write requests and arbitrating them with the WRR scheduler.

[2]The 64-bit counter overflow with updates done every cycle at 1 GHz will happen in $2^{64} \div 10^{-9}$s/op ÷60s ÷60m ÷24h ÷365.25d ≈ 584yrs.

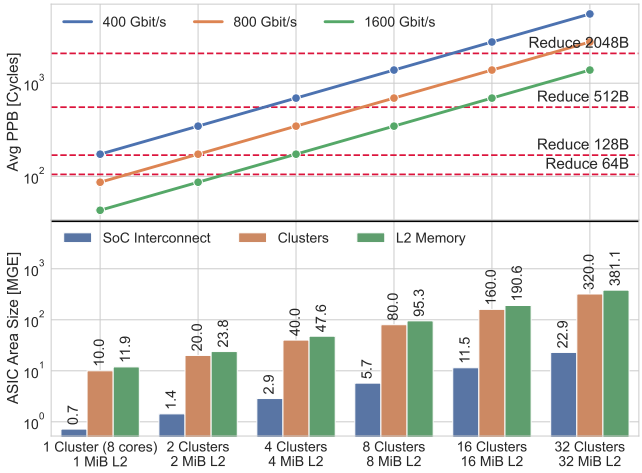

Figure 7: The cost model of sNIC SoC area synthesized in 22nm GF process, compared to the theoretical per packet budget (averaged for different packet sizes at 64 – 4096 B interval) achieved with 400/800/1600 Gbit/s ingress link rates.

6.3 Integration with other on-path SmartNICs

OSMOSIS could be applied to the on-path sNICs designs besides PsPIN. For example, the data path accelerator (DPA) introduced in Bluefield 3 DPU [69, 70] could be extended with OSMOSIS to enable kernel execution QoS. DPA invokes user-defined kernels upon completion of RDMA operations. Thus, FMQ abstraction could be 1:1 mapped to DPA-managed RDMA Completion Queues (CQs) that are arbitrated according to OSMOSIS WLBVT policy. Further, IO operations initiated from DPA cores during kernel execution, i.e., RDMA Work Requests (WRs), could be assigned with a desired Service Level (SL) mapped to the underlying RDMA Virtual Lane (VL), i.e., SL2VL mapping mechanism [41].

7 Evaluation

We study how OSMOSIS allocates sNIC resources under different traffic conditions and workload requirements. We investigate the following research questions:

1. How does the area of OSMOSIS-enabled sNIC chip scale up with the ingress link rates and the number of tenants?
2. What are the overheads of OSMOSIS compared to the reference PsPIN implementation?
3. What is the maximum load that OSMOSIS can sustain?
4. How fair are OSMOSIS resource allocations?

7.1 Hardware Scaling

We synthesized OSMOSIS and PsPIN SystemVerilog IP blocks at 1GHz in GlobalFoundries 22nm node process to es-

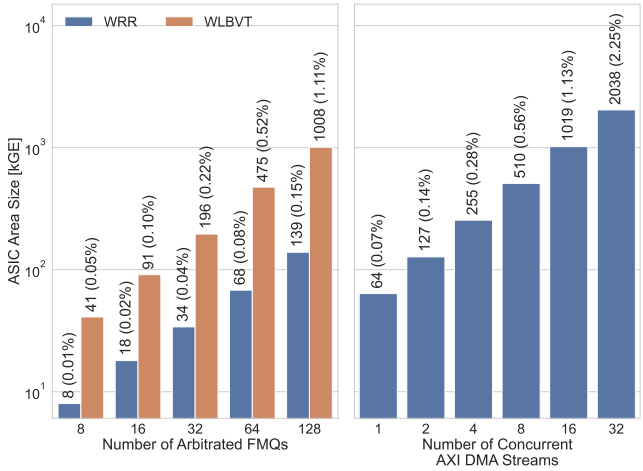

Figure 8: WLBVT and WRR schedulers exhibit linear area scaling in GF 22nm process. Bar captions indicate gate count and relative area compared to 4 PU clusters with 4 MiB L2.

timate hardware area costs using Synopsys Design Compiler NXT in topographic mode.

sNIC area scaling with compute capacity: PsPIN clusters utilize a hierarchical SoC-interconnect similar to Manticore scale-out study [101]. We group four clusters into a *quadrant* sharing a local interconnect. Each quadrant connects to L2 memory, allowing all cores to access the shared packet buffer. Synthesis studies [18, 52] indicate negligible area increases and timing overheads when adding ports to L2. In Figure 7, PsPIN demonstrates linear compute capacity scaling relative to the core area. For instance, 4 PU clusters offer adequate per-packet budget (PPB) (Section 3) to sustain compute-bound Reduce workload with up to 512-byte packets.

OSMOSIS Schedulers Scaling: Figure 8 shows the hardware area consumption of OSMOSIS schedulers. We observe a linear scaling of the FMQ and DMA engine schedulers with the number of inputs. Compared to RR, WLBVT needs 7× more gates, yet with 128 FMQs, WLBVT area consumption takes only 1% of PsPIN cluster and L2 memory area.

7.2 Experimental Methodology

We evaluate OSMOSIS runtime performance using cycle-accurate Verilator v4.228 SystemVerilog simulator [86]. Our experimental testbed features two setups: a *Reference (baseline) PsPIN implementation*, i.e., a conventional on-path sNIC without multi-tenant OS, and a *PsPIN implementation enhanced with OSMOSIS management*.

Both setups feature 4 PsPIN clusters of 8 1GHz cores, achieving 400 Gbit/s ingress/egress bandwidth. L2 and host memories can be accessed through 512 Gbit/s AXI interconnect. We used randomly pre-generated packet traces fully utilizing ingress link bandwidth. Packet arrival sequences follow a uniform distribution, and packet sizes are sampled from a lognormal distribution [10, 81, 97]. For fairness measurements, we use Jain's fairness metric [36]. It scales between 1 and 1/number of tenants: a metric of y implies y% fair treatment, leaving $(100-y)$% starved. Fair treatment ensures equal priority-adjusted resource access for each tenant.

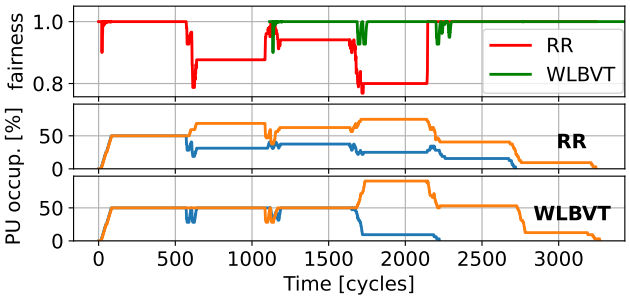

Figure 9: The fairness of WLBVT and RR with two tenants of different compute cost per byte.

7.3 Synthetic Benchmarks

We evaluate OSMOSIS on synthetic benchmarks to assess its overheads in a low-complexity environment.

R1 R5 **Fair HPU allocation:** We evaluate the WLBVT scheduler and compare it to the traditional RR. We run two applications, one with a larger *compute cost per byte*, the *Congestor*, and the other with a smaller one, the *Victim*. Both spin in a `for` loop to simulate a compute-bound task. Figure 9 shows how RR over-allocates PUs to the *Congestor*, leading to lower fairness, as shown by Jain's metric. WLBVT consistently splits all the resources equally between tenants. When the *Victim* has no outstanding packets, WLBVT allows the *Congestor* to overtake more PUs. WLBVT enables fair compute resource allocation within OSMOSIS and does not cause slowdowns within the benchmarks.

R2 R5 **Resolving HoL-blocking:** We evaluate the scaling of throughput of the *Congestor* and the kernel completion time of the *Victim* while conducting only Egress transfers that involve AXI writes. Figure 10 presents how OSMOSIS resolves HoL-blocking. Depending on the fragmentation method, the *Victim*'s kernel completion time can be reduced by order of magnitude while preserving a relative slowdown of only around 2×. The throughput reduction stems from control traffic overhead related to fragmentation. When accessing local sNIC memories (i.e., remote scratchpads and L2), it can be mitigated through a custom SystemVerilog implementation of the PsPIN AXI protocol, allowing for parallel transfer states as proposed in other works [11, 43, 77]. Addressing this issue for host-side traffic that crosses AXI bus boundaries would require a fine-grained QoS protocol for standardized PCIe and CXL interconnects [1].

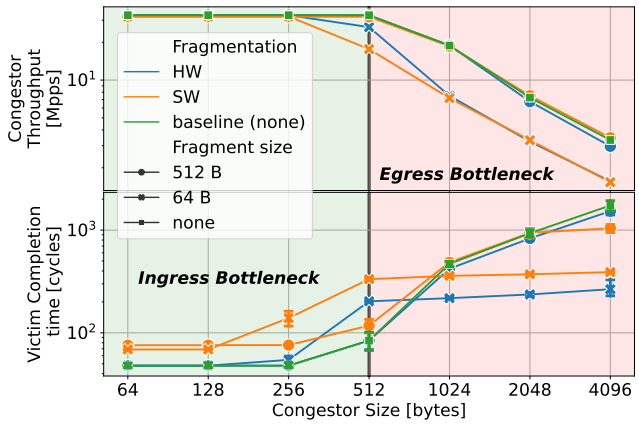

Figure 10: The impact on the *Congestor* throughput and the *Victim* kernel completion time as a function of the *Congestor* size and various fragment sizes.

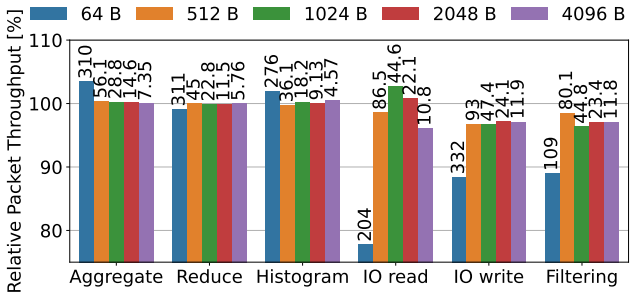

Figure 11: The relative packet throughput of common datacenter workloads run in a standalone mode as a function of packet size with their raw performance in million packets per second (Mpps) at the top of the bars. Up to a 3% throughput increase with OSMOSIS compared to the PsPIN baseline stems from a kernel completion time variability introduced by the compute/IO schedulers.

We also observed two bottlenecks: ingress and egress. In the ingress bottleneck, the incoming link bandwidth is the limit, while in the egress one, the AXI bus congestion causes slowdowns. While the overheads come from the interconnect, OSMOSIS scheduling does not introduce overheads, as evident for low *Congestor* sizes.

7.4 Datacenter Workloads

Additionally, we evaluate a set of real datacenter workloads supplied with the PsPIN benchmarking package [18]. We study the *Aggregation* [71], *Reduction* [9] and *Histogram* [7] benchmarks as examples of compute-bound workloads with incrementally increasing inter-kernel memory synchronization requirements, i.e., from local on-PU computation with one atomic operation in *Aggregation*, to random memory accesses, each with an atomic summation in *Histogram*.

We also evaluate an IO-bound benchmarking set. Our goal is to exercise NIC DMA read/write data paths towards the host memory, the pattern typical for data path offloading of storage RPCs and TCP segment delivery [62,65,74,83]. While for *IO read/write*, a target memory location is stored directly in the packet application header, in the *Filtering* benchmark, to lookup the destination DMA memory address (e.g., KVS-cache location or packet forwarding table context address), the kernel needs to compute the hash of the L7-header used as a lookup table index stored in sNIC LLC.

Management overheads: To assess the influence of OSMOSIS management on applications' performance, we start by running them in isolation. Figure 11 displays how OSMOSIS does not introduce considerable overheads for compute-bound workloads. These oscillate within ±3% of the baseline PsPIN implementation and reach the maximum of 310Mpps for the *Aggregation* workloads. For IO-bound workloads, OSMOSIS introduces overheads stemming from the fragmentation, which have been discussed in Section 7.3. This can be resolved by extending the AXI bus protocol [43, 77]. While overheads reach from 23% to 2% and represent the cost of introducing fair and efficient multi-tenancy, the workloads still achieve 332Mpps in the *IO write* case.

Application mixtures: Evaluating applications in isolation is not representative of real workloads which occur in multi-tenant datacenters for which OSMOSIS was designed and where multiple users contend for resources. We consider two application sets: a *compute-bound set* and an *IO-bound* set, each resulting in tenant resource contention.

The compute-bound set comprises the *Reduce* and *Histogram* workloads. Each is introduced as a *Victim* (64B packets for *Reduce* and 64-128 packets for *Histogram*) and *Congestor* (4KB packets for *Reduce* and 3072 – 4096 byte packets for *Histogram*). As Figure 12 shows, these workloads saturate the PUs of the sNIC within the first couple thousand cycles and introduce compute congestion. Using OSMOSIS WLBVT scheduling, each tenant obtains an allocation that is, on average, 47% fairer than that of the typical RR implementation as measured using Jain's metric. Such allocations ensure SLO fulfillment and result in 39-7% faster *flow completion times* (FCT) because of lower average contention while only sacrificing 3% of the *Histogram Congestor*. OSMOSIS thus achieves a fair and efficient resource allocation.

The IO-bound set consists of *IO read* and *write* workloads which are again introduced as both a *Victim* and *Congestor* with the same packet size parameters as the *Histogram* workload. For the IO-bound workloads, we focus on the average throughput of each workload. Figure 13 shows that, similarly to the compute case, OSMOSIS obtains a consistently fairer allocation than a traditional RR scheduler (up to 83%) as measured by the average Jain's fairness metric.

OSMOSIS also manages to reduce FCT for all tenants by

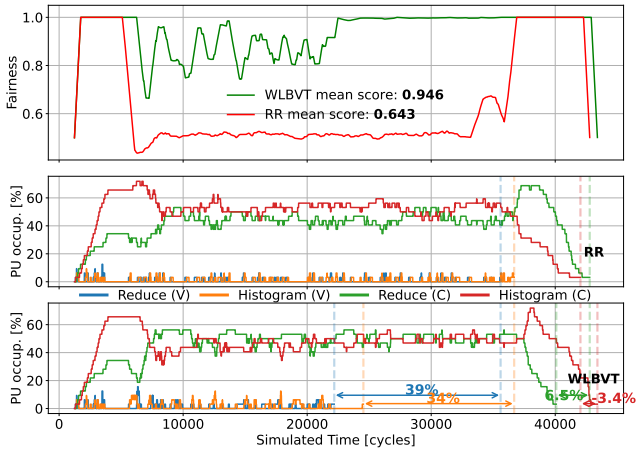

Figure 12: The evolution of tenant PU occupation and time average fairness against the simulated time in cycles. The percentages indicate the reduction in FCT for each tenant.

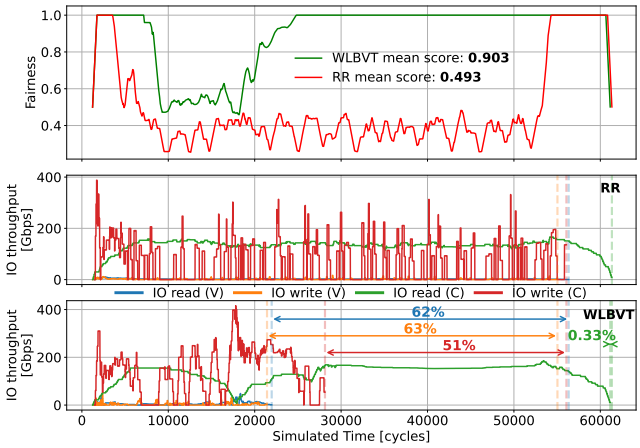

Figure 13: The evolution of tenant IO throughput and time average fairness against the simulated time in cycles. The percentages indicate the reduction in FCT for each tenant.

up to 63%. Such large improvement comes from addressing the HoL-blocking problem, leading to a more efficient allocation. The *IO read Congestor* is initially suppressed to let other tenants fairly finish their workloads and then obtains full exclusive utilization, eliminating contention and allowing it to regain the lost performance. On the other hand, the other tenants are fairly allocated and, as Figure 14 shows, they do not suffer from HoL-blocking.

Figure 14 also displays the true cost of the aforementioned gains. While the overall FCT is reduced for all tenants, the single kernel completion time shows a different story. The HoL-blocking is resolved for the *Victim* tenants, for which the kernel completion time is reduced more than fivefold. However, the other *Congestor* tenants display an up to eightfold increased median kernel completion time. While OSMOSIS

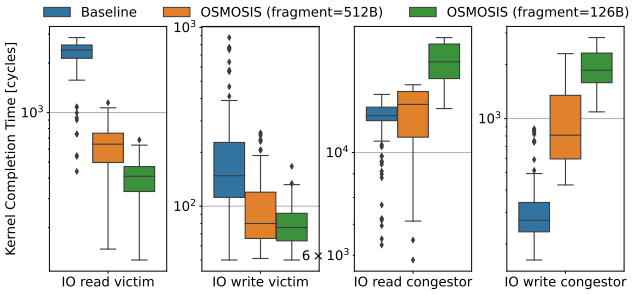

Figure 14: The completion time distribution for IO-bound applications for two fragment sizes.

increases the median single-packet processing time, it also achieves overall FCT gains for the IO set by allocating the resources fairly and more efficiently, and by parallelizing the packets appropriately.

8 Conclusions

Enabling user-level on-NIC processing in modern multi-tenant datacenters brings resource multiplexing challenges. OSMOSIS solves sNIC multi-tenancy by distributing sNIC resources, the egress and DMA bandwidth, and processing units, across flows with different priorities, input bandwidth, and computational requirements. To achieve a fair distribution of resources, OSMOSIS relies on sNIC-specific principles, such as work-conservative allocation of compute and IO resources. The evaluation shows that OSMOSIS efficiently redistributes resources, enabling QoS, performance isolation, and prioritization between various mixtures of flows. OSMOSIS improves FCT by up to 60% and is fairer by up to 83% than the typical schedulers. We believe that OSMOSIS could enable wider adoption of sNICs in cloud datacenters with low overhead.

Acknowledgments

This project received funding from EuroHPC-JU under the grant agreements RED-SEA, No. 055776 and DEEP-SEA, No. 95560, the EuroHPC-JU "The European Pilot" project under the grant agreement No. 101034126 as part of the EU Horizon 2020 research and innovation programme, and a donation from Intel.

References

[1] Agarwal, S., Agarwal, R., Montazeri, B., Moshref, M., Elmeleegy, K., Rizzo, L., de Kruijf, M. A., Kumar, G., Ratnasamy, S., Culler, D., et al. Understanding host interconnect congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (2022), pp. 198–204.

[2] AGRAWAL, A., AND KIM, C. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)* (2020), IEEE Computer Society, pp. 1–32.

[3] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference* (2010), pp. 63–74.

[4] ANDERSON, T. E., OWICKI, S. S., SAXE, J. B., AND THACKER, C. P. High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems (TOCS) 11*, 4 (1993), 319–352.

[5] ATTIG, M., AND BREBNER, G. 400 gb/s programmable packet parsing on a single fpga. In *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems* (2011), IEEE, pp. 12–23.

[6] BALAS, R., AND BENINI, L. Risc-v for real-time mcus - software optimization and microarchitectural gap analysis. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2021), pp. 874–877.

[7] BARTHELS, C., MÜLLER, I., SCHNEIDER, T., ALONSO, G., AND HOEFLER, T. Distributed join algorithms on thousands of cores. *Proceedings of the VLDB Endowment 10*, 5 (2017), 517–528.

[8] BELAY, A., PREKAS, G., KLIMOVIC, A., GROSSMAN, S., KOZYRAKIS, C., AND BUGNION, E. {IX}: a protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 49–65.

[9] BEN-NUN, T., AND HOEFLER, T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR) 52*, 4 (2019), 1–43.

[10] BENSON, T., AKELLA, A., AND MALTZ, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC '10, Association for Computing Machinery, p. 267–280.

[11] BENZ, T., ROGENMOSER, M., SCHEFFLER, P., RIEDEL, S., OTTAVIANO, A., KURTH, A., HOEFLER, T., AND BENINI, L. A high-performance, energy-efficient modular dma engine architecture. *arXiv preprint arXiv:2305.05240* (2023).

[12] BLÖCHER, M., WANG, L., EUGSTER, P., AND SCHMIDT, M. Switches for hire: Resource scheduling for data center in-network computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), pp. 268–285.

[13] BOLCH, G., GREINER, S., DE MEER, H., AND TRIVEDI, K. S. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, USA, 1998.

[14] BORROMEO, J. C., KONDEPU, K., ANDRIOLLI, N., AND VALCARENGHI, L. Fpga-accelerated smartnic for supporting 5g virtualized radio access network. *Computer Networks 210* (2022), 108931.

[15] CAI, Q., VUPPALAPATI, M., HWANG, J., KOZYRAKIS, C., AND AGARWAL, R. Towards μ s tail latency and terabit ethernet: disaggregating the host network stack. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 767–779.

[16] CHANG, B., AKELLA, A., D'ANTONI, L., AND SUBRAMANIAN, K. Learned load balancing. In *Proceedings of the 24th International Conference on Distributed Computing and Networking* (2023), pp. 177–187.

[17] DEAN, J., AND BARROSO, L. A. The tail at scale. *Communications of the ACM 56*, 2 (2013), 74–80.

[18] DI GIROLAMO, S., KURTH, A., CALOTOIU, A., BENZ, T., SCHNEIDER, T., BERANEK, J., BENINI, L., AND HOEFLER, T. A risc-v in-network accelerator for flexible high-performance low-power packet processing. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)* (2021), IEEE, pp. 958–971.

[19] DONG, Y., YANG, X., LI, J., LIAO, G., TIAN, K., AND GUAN, H. High performance network virtualization with sr-iov. *Journal of Parallel and Distributed Computing 72*, 11 (2012), 1471–1480.

[20] DONG, Y., YU, Z., AND ROSE, G. Sr-iov networking in xen: Architecture, design and implementation. In *Workshop on I/O virtualization* (2008), vol. 2.

[21] DUDA, K. J., AND CHERITON, D. R. Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. *ACM SIGOPS Operating Systems Review 33*, 5 (1999), 261–276.

[22] ERAN, H., FUDIM, M., MALKA, G., SHALOM, G., COHEN, N., HERMONY, A., LEVI, D., LISS, L., AND SILBERSTEIN, M. Flexdriver: A network driver for your accelerator. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2022), pp. 1115–1129.

[23] ETHERNET ALLIANCE. Ethernet Roadmap 2022. https://ethernetalliance.org/technology/ethernet-roadmap/.

[24] FARSHIN, A., BARBETTE, T., ROOZBEH, A., MAGUIRE JR, G. Q., AND KOSTIĆ, D. Packetmill: toward per-core 100-gbps networking. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), pp. 1–17.

[25] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking 1*, 4 (1993), 397–413.

[26] FRIED, J., RUAN, Z., OUSTERHOUT, A., AND BELAY, A. Caladan: Mitigating interference at microsecond timescales. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation* (2020), pp. 281–297.

[27] GAO, P., DALLEGGIO, A., XU, Y., AND CHAO, H. J. Gearbox: A hierarchical packet scheduler for approximate weighted fair queuing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 551–565.

[28] GAO, P. X., NARAYAN, A., KARANDIKAR, S., CARREIRA, J., HAN, S., AGARWAL, R., RATNASAMY, S., AND SHENKER, S. Network requirements for resource disaggregation. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)* (2016), pp. 249–264.

[29] GRANT, S., YELAM, A., BLAND, M., AND SNOEREN, A. C. Smartnic performance isolation with fairnic: Programmable networking for the cloud. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 681–693.

[30] GUO, Z., SHAN, Y., LUO, X., HUANG, Y., AND ZHANG, Y. Clio: A hardware-software co-designed disaggregated memory system. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2022), pp. 417–433.

[31] HAECKI, R., MYSORE, R. N., SURESH, L., ZELLWEGER, G., GAN, B., MERRIFIELD, T., BANERJEE, S., AND ROSCOE, T. How to diagnose nanosecond network latencies in rich end-host stacks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 861–877.

[32] HOEFLER, T., DI GIROLAMO, S., TARANOV, K., GRANT, R. E., AND BRIGHTWELL, R. spin: High-performance streaming processing in the network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), pp. 1–16.

[33] HOEFLER, T., ROWETH, D., UNDERWOOD, K., ALVERSON, R., GRISWOLD, M., TABATABAEE, V., KALKUNTE, M., ANUBOLU, S., SHEN, S., MCLAREN, M., ET AL. Data center ethernet and remote direct memory access: Issues at hyperscale. *Computer 56*, 7 (2023), 67–77.

[34] HOFEMEIER, G., AND CHESEBROUGH, R. Introduction to intel aes-ni and intel secure key instructions. *Intel, White Paper 62* (2012).

[35] HØILAND-JØRGENSEN, T., BROUER, J. D., BORKMANN, D., FASTABEND, J., HERBERT, T., AHERN, D., AND MILLER, D. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies* (2018), pp. 54–66.

[36] HOSSFELD, T., SKORIN-KAPOV, L., HEEGAARD, P. E., AND VARELA, M. Definition of qoe fairness in shared systems. *IEEE Communications Letters 21*, 1 (2016), 184–187.

[37] HUNTER, A., KENNELLY, C., TURNER, P., GOVE, D., MOSELEY, T., AND RANGANATHAN, P. Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* (July 2021), USENIX Association, pp. 257–273.

[38] IBANEZ, S., MALLERY, A., ARSLAN, S., JEPSEN, T., SHAHBAZ, M., KIM, C., AND MCKEOWN, N. The nanopu: A nanosecond network stack for datacenters. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)* (2021), pp. 239–256.

[39] IBANEZ, S., MALLERY, A., ARSLAN, S., JEPSEN, T., SHAHBAZ, M., KIM, C., AND MCKEOWN, N. Enabling the reflex plane with the nanopu. *arXiv preprint arXiv:2212.06658* (2022).

[40] IEEE. 802.1Qbb: IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks – Amendment: Priority-based Flow Control. https://1.ieee802.org/dcb/802-1qbb/.

[41] INFINIBAND TRADE ASSOCIATION. InfiniBand Specification. https://www.infinibandta.org.

[42] IVANOV, A., DRYDEN, N., BEN-NUN, T., LI, S., AND HOEFLER, T. Data movement is all you need: A case study on optimizing transformers. *Proceedings of Machine Learning and Systems 3* (2021), 711–732.

[43] JIANG, Z., YANG, K., FISHER, N., GRAY, I., AUDSLEY, N. C., AND DONG, Z. Axi-ic rt: Towards a real-time axi-interconnect for highly integrated socs. *IEEE Transactions on Computers 72*, 3 (2022), 786–799.

[44] KAFFES, K., CHONG, T., HUMPHRIES, J. T., BELAY, A., MAZIÈRES, D., AND KOZYRAKIS, C. Shinjuku: Preemptive scheduling for {μsecond-scale} tail latency. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)* (2019), pp. 345–360.

[45] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (Denver, CO, June 2016), USENIX Association, pp. 437–450.

[46] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Datacenter rpcs can be general and fast. *arXiv preprint arXiv:1806.00680* (2018).

[47] KATSIKAS, G. P., BARBETTE, T., CHIESA, M., KOSTIĆ, D., AND MAGUIRE JR, G. Q. What you need to know about (smart) network interface cards. In *International Conference on Passive and Active Network Measurement* (2021), Springer, pp. 319–336.

[48] KHAWAJA, A., LANDGRAF, J., PRAKASH, R., WEI, M., SCHKUFZA, E., AND ROSSBACH, C. J. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)* (2018), pp. 107–127.

[49] KIM, J., JANG, I., REDA, W., IM, J., CANINI, M., KOSTIĆ, D., KWON, Y., PETER, S., AND WITCHEL, E. Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (2021), pp. 756–771.

[50] KOROLIJA, D., ROSCOE, T., AND ALONSO, G. Do os abstractions make sense on fpgas? In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation* (2020), pp. 991–1010.

[51] KUMAR, P., DUKKIPATI, N., LEWIS, N., CUI, Y., WANG, Y., LI, C., VALANCIUS, V., ADRIAENS, J., GRIBBLE, S., FOSTER, N., ET AL. Picnic: predictable virtualized nic. In *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 351–366.

[52] KURTH, A., RÖNNINGER, W., BENZ, T., CAVALCANTE, M., SCHUIKI, F., ZARUBA, F., AND BENINI, L. An open-source platform for high-performance non-coherent on-chip communication. *IEEE Transactions on Computers 71*, 8 (2021), 1794–1809.

[53] LAZAREV, N., XIANG, S., ADIT, N., ZHANG, Z., AND DELIMITROU, C. Dagger: efficient and fast rpcs in cloud microservices with near-memory reconfigurable nics. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), pp. 36–51.

[54] LE, Y., CHANG, H., MUKHERJEE, S., WANG, L., AKELLA, A., SWIFT, M. M., AND LAKSHMAN, T. Uno: Uniflying host and smart nic offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), pp. 506–519.

[55] LI, Y., MIAO, R., LIU, H. H., ZHUANG, Y., FENG, F., TANG, L., CAO, Z., ZHANG, M., KELLY, F., ALIZADEH, M., ET AL. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 44–58.

[56] LIN, J., PATEL, K., STEPHENS, B. E., SIVARAMAN, A., AND AKELLA, A. PANIC: A High-Performance programmable NIC for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (Nov. 2020), USENIX Association, pp. 243–259.

[57] LIU, M., CUI, T., SCHUH, H., KRISHNAMURTHY, A., PETER, S., AND GUPTA, K. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 318–333.

[58] LIU, M., PETER, S., KRISHNAMURTHY, A., AND PHOTHILIMTHANA, P. M. E3: Energy-efficient microservices on smartnic-accelerated servers. In *USENIX annual technical conference* (2019), pp. 363–378.

[59] MARVELL. LiquidIO-III. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf.

[60] MIANO, S., SANAEE, A., RISSO, F., RÉTVÁRI, G., AND ANTICHI, G. Domain specific run time optimization for software data planes. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2022), pp. 1148–1164.

[61] MICROSOFT. Introduction to Receive Side Scaling. https://learn.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-receive-side-scaling.

[62] MIN, J., LIU, M., CHUGH, T., ZHAO, C., WEI, A., DOH, I. H., AND KRISHNAMURTHY, A. Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (2021), pp. 106–122.

[63] MINTURN, D. Nvm express over fabrics. In *11th Annual OpenFabrics International OFS Developers' Workshop* (2015).

[64] MOGUL, J. C. Tcp offload is a dumb idea whose time has come. In *HotOS* (2003), pp. 25–30.

[65] MOON, Y., LEE, S., JAMSHED, M. A., AND PARK, K. {AccelTCP}: Accelerating network applications with stateful {TCP} offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 77–92.

[66] MUDIGONDA, J., YALAGANDULA, P., MOGUL, J., STIEKES, B., AND POUFFARY, Y. Netlord: a scalable multi-tenant network architecture for virtualized datacenters. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 62–73.

[67] NAMYAR, P., SUPITTAYAPORNPONG, S., ZHANG, M., YU, M., AND GOVINDAN, R. A throughput-centric view of the performance of datacenter topologies. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (2021), pp. 349–369.

[68] NETRONOME. Agilio SmartNICs. https://www.netronome.com/products/smartnic/overview/.

[69] NVIDIA. Bluefield-3 DPA FlexIO. https://docs.nvidia.com/doca/sdk/dpa-subsystem-programming-guide/index.html.

[70] NVIDIA. Bluefield-3 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf.

[71] ORDONEZ, C., AND CHEN, Z. Horizontal aggregations in sql to prepare data sets for data mining analysis. *IEEE transactions on knowledge and data engineering 24*, 4 (2011), 678–691.

[72] PETER, S., LI, J., ZHANG, I., PORTS, D. R., WOOS, D., KRISHNAMURTHY, A., ANDERSON, T., AND ROSCOE, T. Arrakis: The operating system is the control plane. *ACM Transactions on Computer Systems (TOCS) 33*, 4 (2015), 1–30.

[73] PHOTHILIMTHANA, P. M., LIU, M., KAUFMANN, A., PETER, S., BODIK, R., AND ANDERSON, T. Floem: A programming system for NIC-Accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 663–679.

[74] PISMENNY, B., ERAN, H., YEHEZKEL, A., LISS, L., MORRISON, A., AND TSAFRIR, D. Autonomous nic offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), pp. 18–35.

[75] POURHABIBI, A., SUTHERLAND, M., DAGLIS, A., AND FALSAFI, B. Cerebros: Evading the rpc tax in datacenters. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (2021), pp. 407–420.

[76] PREKAS, G., KOGIAS, M., AND BUGNION, E. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), pp. 325–341.

[77] RESTUCCIA, F., BIONDI, A., MARINONI, M., CICERO, G., AND BUTTAZZO, G. Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc. In *2020 57th ACM/IEEE Design Automation Conference (DAC)* (2020), IEEE, pp. 1–6.

[78] RESTUCCIA, F., PAGANI, M., BIONDI, A., MARINONI, M., AND BUTTAZZO, G. Is your bus arbiter really fair? restoring fairness in axi interconnects for fpga socs. *ACM Transactions on Embedded Computing Systems (TECS) 18*, 5s (2019), 1–22.

[79] RIVITTI, A., BIFULCO, R., TULUMELLO, A., BONOLA, M., AND PONTARELLI, S. ehdl: Turning ebpf/xdp programs into hardware designs for the nic. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (2023), pp. 208–223.

[80] ROSSI, D., CONTI, F., MARONGIU, A., PULLINI, A., LOI, I., GAUTSCHI, M., TAGLIAVINI, G., CAPOTONDI, A., FLATRESSE, P., AND BENINI, L. Pulp: A parallel ultra low power platform for next generation iot applications. In *2015 IEEE Hot Chips 27 Symposium (HCS)* (2015), IEEE Computer Society, pp. 1–39.

[81] ROY, A., ZENG, H., BAGGA, J., PORTER, G., AND SNOEREN, A. C. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev. 45*, 4 (Aug. 2015), 123–137.

[82] SEYEDROUDBARI, H., VANAVASAM, S., AND DAGLIS, A. Turbo: Smartnic-enabled dynamic load balancing of μs-scale rpcs. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2023), IEEE, pp. 1045–1058.

[83] SHASHIDHARA, R., STAMLER, T., KAUFMANN, A., AND PETER, S. {FlexTOE}: Flexible {TCP} offload with {Fine-Grained} parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 87–102.

[84] SINGLA, A., GODFREY, P. B., AND KOLLA, A. High throughput data center topology design. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (2014), pp. 29–41.

[85] SIRACUSANO, G., GALEA, S., SANVITO, D., MALEKZADEH, M., ANTICHI, G., COSTA, P., HADDADI, H., AND BIFULCO, R. Re-architecting traffic analysis with neural network interface cards. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (Renton, WA, Apr. 2022), USENIX Association, pp. 513–533.

[86] SNYDER, W. Verilator: Open simulation-growing up. *DVClub Bristol* (2013).

[87] STEPHENS, B. E., AKELLA, A., AND SWIFT, M. M. Loom: Flexible and efficient nic packet scheduling. In *NSDI* (2019), vol. 19, pp. 33–46.

[88] SUN, S., ZHANG, R., YAN, M., AND WU, J. Skv: A smartnic-offloaded distributed key-value store. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)* (2022), IEEE, pp. 1–11.

[89] SWAMY, T., RUCKER, A., SHAHBAZ, M., GAUR, I., AND OLUKOTUN, K. Taurus: a data plane architecture for per-packet ml. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2022), pp. 1099–1114.

[90] TORK, M., MAUDLEJ, L., AND SILBERSTEIN, M. Lynx: A smartnic-driven accelerator-centric architecture for network servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (2020), pp. 117–131.

[91] ULTRAETHERNET CONSORTIUM. The New Era Needs a New Network. https://ultraethernet.org.

[92] VAHDAT, A., AND MILOJICIC, D. The next wave in cloud systems architecture. *Computer 54*, 10 (2021), 116–120.

[93] WANG, T., YANG, X., ANTICHI, G., SIVARAMAN, A., AND PANDA, A. Isolation mechanisms for High-Speed Packet-Processing pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (Renton, WA, Apr. 2022), USENIX Association, pp. 1289–1305.

[94] WANG, Z., HUANG, H., ZHANG, J., WU, F., AND ALONSO, G. {FpgaNIC}: An {FPGA-based} versatile 100gb {SmartNIC} for {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)* (2022), pp. 967–986.

[95] WATERMAN, A., LEE, Y., AVIZIENIS, R., PATTERSON, D. A., AND ASANOVIĆ, K. The risc-v instruction set manual volume ii: Privileged architecture version 1.9. Tech. Rep. UCB/EECS-2016-129, EECS Department, University of California, Berkeley, Jul 2016.

[96] WEIBAI, X. J., XU, Y., ELHADDAD, M., RAINDEL, S., PADHYE, J., AND ZHUO, A. R. L. D. Understanding rdma microarchitecture resources for performance isolation.

[97] WOODRUFF, J., MOORE, A. W., AND ZILBERMAN, N. Measuring burstiness in data center applications. In *Proceedings of the 2019 Workshop on Buffer Sizing* (New York, NY, USA, 2019), BS '19, Association for Computing Machinery.

[98] YAN, Y., BELDACHI, A. F., NEJABATI, R., AND SIMEONIDOU, D. P4-enabled smart nic: Enabling sliceable and service-driven optical data centres. *Journal of Lightwave Technology 38*, 9 (2020), 2688–2694.

[99] YANG, X., EGGERT, L., OTT, J., UHLIG, S., SUN, Z., AND ANTICHI, G. Making quic quicker with nic offload. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (2020), pp. 21–27.

[100] YUAN, Y., HUANG, J., SUN, Y., WANG, T., NELSON, J., PORTS, D. R., WANG, Y., WANG, R., TAI, C., AND KIM, N. S. Rambda: Rdma-driven acceleration framework for memory-intensive μs-scale datacenter applications. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2023), IEEE, pp. 499–515.

[101] ZARUBA, F., SCHUIKI, F., AND BENINI, L. Manticore: A 4096-core risc-v chiplet architecture for ultraefficient floating-point computing. *IEEE Micro 41*, 2 (2020), 36–42.

[102] ZHANG, Y., TAN, Y., STEPHENS, B., AND CHOWDHURY, M. Justitia: Software Multi-Tenancy in hardware Kernel-Bypass networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (Renton, WA, Apr. 2022), USENIX Association, pp. 1307–1326.

[103] ZHU, Y., ERAN, H., FIRESTONE, D., GUO, C., LIPSHTEYN, M., LIRON, Y., PADHYE, J., RAINDEL, S., YAHIA, M. H., AND ZHANG, M. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 523–536.