Not A DPU in Name Only! Unleashing RDMA-capable DPUs in Multi-Tenant Serverless Clouds with NADINO

Shixiong Qi[△] Songyu Zhang[†] K. K. Ramakrishnan[†] Diman Z. Tootaghaj[⋆] Hardik Soni[⋆] Puneet Sharma[⋆]
[△]University of Kentucky [†]University of California, Riverside [⋆]Hewlett Packard Labs

Abstract

Serverless computing promises enhanced resource efficiency and lower user costs, yet is burdened by a heavyweight, CPU-bound data plane. Prior efforts exploiting shared memory reduce overhead locally but fall short when scaling across nodes. Furthermore, serverless environments can have unpredictable and large-scale multi-tenancy, leading to contention for shared network resources.

We present NADINO, a DPU-centric serverless data plane that reduces the CPU burden and enables efficient, zero-copy communication in multi-tenant serverless clouds. Despite the limited general-purpose processing capability of the DPU cores, NADINO strategically exploits the DPU's potential by (1) offloading data transmission to high-performance NIC cores via RDMA, combined with intra-node shared memory to eliminate data copies across nodes, and (2) enabling cross-processor (CPU-DPU) shared memory to eliminate redundant data movement, which overwhelms wimpy DPU cores. At the core of NADINO is the DPU-enabled network engine (DNE) – a lightweight reverse proxy that isolates RDMA resources from tenant functions, orchestrates inter-node RDMA flows, and enforces fairness under contention.

To further reduce CPU involvement, NADINO performs early HTTP/TCP-to-RDMA transport conversion at the cloud ingress, bridging the protocol mismatch before client traffic enters the RDMA fabric, thus avoiding costly protocol translation along the critical path. We show that careful selection of RDMA primitives (i.e., two-sided instead of one-sided) significantly affects the zero-copy data plane.

Our preliminary experimental results show that enabling DPU offloading in NADINO improves RPS by 20.9×. The latency is reduced by a factor of 21× in the best case, all the while saving up to 7 CPU cores, and only consuming two wimpy DPU cores.

CCS Concepts: • **Networks → Cloud computing**; **Data center networks**; • **Computer systems organization → Cloud computing**.

Keywords: DPU, RDMA, Multi-tenancy, Serverless

EUROSYS '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2212-7/26/04
https://doi.org/10.1145/3767295.3769386

ACM Reference Format:
S. Qi, S. Zhang, K. K. Ramakrishnan, D. Z. Tootaghaj, H. Soni, P. Sharma. 2026. Not A DPU in Name Only! Unleashing RDMA-capable DPUs in Multi-Tenant Serverless Clouds with NADINO. In *21st European Conference on Computer Systems (EUROSYS '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3767295.3769386

1 Introduction

Serverless computing, or Function-as-a-Service (FaaS [7]), eases the burden on application developers to provision and manage cloud resources, while its fine-grained resource elasticity dramatically reduces user costs [44, 80, 92]. Despite many positive features, the data plane in current serverless platforms is heavily *CPU-bound*, caused by significant overheads from kernel-based inter-function networking [23, 42, 56, 78]. This cost is further amplified by the loose coupling between serverless functions, resulting in duplicate data plane processing in a "chain" of disaggregated functions and severely impacting performance.

To reduce the data plane overhead in serverless environments, existing solutions have leveraged shared memory processing, enabling faster and efficient *zero-copy* communication between functions [29, 43, 78, 100]. However, they are primarily for intra-node communication, restricting their applicability in a distributed serverless system where functions of a chain may be spread across multiple nodes. RDMA-based network fabrics, widely deployed in data centers [20, 40], can extend zero-copy communication[1] to distributed environments and enable low-latency, efficient inter-function communication by leveraging hardware acceleration [32, 64, 67]. More recently, RDMA NICs (RNICs) have been enhanced with onboard, general-purpose processing cores, evolving into SmartNIC-like Data Processing Units (DPUs) [6, 95]. DPUs can offload and orchestrate data plane tasks directly on the integrated RNICs and have become a fundamental element of compute nodes in the cloud [8, 9, 35, 69, 86]. Despite these infrastructure-level advancements, several challenges remain to be addressed before its full potential can be unleashed in serverless environments:

Multi-tenancy related challenges persist in the serverless environment, with resource contention becoming even more

[1]In our context, "zero-copy" specifically refers to the elimination of software-based data copies, while allowing hardware-based mechanisms like DMA/RDMA for efficient data movement.

dynamic and unpredictable. This stems from the fundamental goal of fine-grained resource elasticity of serverless computing. Frequent configuration changes due to workload variation, function placement and auto-scaling require corresponding flexibility in provisioning of compute/network resources for each tenant. While existing platforms have developed robust CPU, memory, and traditional network I/O isolation across tenants [84, 89, 90], they largely overlook RDMA-related network isolation, leaving inter-tenant RDMA traffic vulnerable to interference. This gap exists because serverless functions typically lack the privilege to access RNIC directly (i.e., operating RDMA Queue Pairs (QPs) via IB verbs). Granting untrusted user functions direct access to RDMA resources can introduce significant security risks [41, 48, 91] and fail to ensure fair sharing (*e.g.,* of the bandwidth of the RDMA Fabric) among different tenants (more details in §2.1).

Serverless platforms strive to maximize the density of user functions per node [19, 60]. However, having the CPU, which is the primary place for function execution, perform auxiliary orchestration tasks (such as managing multi-tenancy and operating RDMA QPs) consumes valuable CPU cycles, which instead could be used for executing serverless functions. DPUs present an attractive offloading target, but face two principal limitations: *(1) Wimpy DPU cores:* DPU cores are generally much less powerful than those of CPUs, so they struggle with heavyweight packet processing [65, 69], especially for protocol processing and state management on the DPU. *(2) Cross-processor (CPU-DPU) data movement overhead:* To perform network orchestration on the DPU, inbound packets must traverse from the RNIC to the DPU, before onward to the CPU for function execution, and vice versa for outbound packets. This "on-path" CPU–DPU data movement incurs nontrivial overhead, despite being accelerated by the DPU's DMA engine. Empirical measurements (see §4.1.1) show up to a 1.54× performance degradation, compared to having the data directly transmitted from host (CPU) out to the network using the RNIC's DMA.

To create a zero-copy dataplane in a distributed environment, inter-node (RDMA) and intra-node (shared memory) data transfers should share a *unified memory pool.* Maintaining separate pools for inter-node and intra-node communication inevitably results in data copies (Fig. 3 (2)). But sharing a unified memory pool for a combined inter-node and intra-node dataplane is nontrivial. Since data buffers are shared, careful management of data ownership is essential to prevent contention. Although locks and collective synchronization (such as Rendezvous [88]) can be used to manage data ownership between distributed functions, they can also significantly degrade performance (Fig. 3 (1)).

Additionally, external clients use HTTP/TCP while the RDMA Fabric uses a more lightweight transport. Existing RDMA-based serverless data planes often overlook this transport mismatch, still relying on HTTP/TCP ingress gateways at the cluster edge to forward TCP connection traffic from external clients to worker nodes. The worker's local TCP/IP stack has to process requests before switching to RDMA or shared memory (see Fig. 4). This redundant processing at both the ingress and worker nodes leads to inefficiencies. Our findings (§4.1.3) show up to 11.4× performance degradation and also significant CPU consumption. We believe the TCP connection ought to be properly terminated at the ingress gateway, and only the payload efficiently transferred over RDMA to maximize performance.

NADINO Overview. To address these concerns, we propose NADINO, a DPU-centric serverless data plane that seamlessly integrates intra-node shared memory processing with inter-node RDMA-based data transfers, interacting over a unified memory pool on each worker node. This facilitates *true* zero-copy data transfers between functions and eliminates the heavyweight kernel-based data transfers, regardless of their physical location within the serverless cloud.

To support multi-tenancy in serverless environments using RDMA Fabrics, NADINO introduces a node-wide, DPU-enabled Network Engine (called "DNE") as a software indirection layer to operate RDMA QPs on behalf of user functions. This allows fast inter-node communication while isolating the RNIC from direct access by untrusted functions. This centralized management of a node's RDMA resources enforces per-tenant traffic management policies, ensuring fair sharing of RDMA resources (*e.g.,* bandwidth) among tenants (§3.3). In addition, DPU offloading offers two key benefits: (1) isolating the trusted network engine *physically* from untrusted user functions running on the CPU [16, 25, 75] to maintain its integrity, while (2) mitigating any CPU consumption for executing network engine tasks.

Unleashing DPU Offloading with NADINO. To unleash the power of the DPU, despite its wimpy processor cores, NADINO incorporates several key innovations into the DNE: *(1) Cross-processor (CPU-DPU) memory sharing* eliminates the need (and associated cost) for "on-path" data movement by the DNE (§3.4.2). This further enables the split of control and data path and ensures that each component on DPU is used for what it is best suited to: The DNE serves purely as an *off-path* controller, leveraging the programmable DPU cores for managing the RDMA data transfers and ensuring fair sharing of RNIC resources across tenants, while the RNIC itself handles the high-performance data plane transfers using the DMA-based data movement. This is similar in design to [104]. *(2) Run-to-completion packet processing* (§3.2). The DNE employs a non-blocking, run-to-completion I/O model to handle two-sided RDMA operations and support multi-tenancy, inspired by the approach taken by IX [22] and demikernel [102]. This minimizes context switching and CPU scheduling overheads, alleviating the performance constraints of the wimpy DPU cores.

To address both the memory management & synchronization challenge and the mismatch between HTTP/TCP vs.

RDMA transport protocol mismatch, we introduce two key innovations in NADINO's zero-copy data plane: *(1) Lock-free communication primitives* (§3.5). In serverless environments, functions interact through a natural producer–consumer relationship. NADINO exploits this to adopt two-sided RDMA primitives for inter-node communication. Since the consumer always posts receive buffers in advance, the producer can issue sends without requiring distributed locks or incurring additional copies by the consumer. On the other hand, the copy would be inevitable when using one-sided RDMA (see discussion in §2.1). In intra-node data plane, we adopt a 'token-passing' scheme for ownership transfer of shared memory buffers. Both mitigate the need of locks and avoid the heavyweight synchronization schemes like Rendezvous. *(2) Early transport conversion at cluster ingress* (§3.6). By performing HTTP/TCP-to-RDMA conversion at the cluster-wide ingress (the earliest entry point into the serverless cloud), NADINO removes all software-based protocol processing overheads from the worker nodes. This aligns perfectly with NADINO's zero-copy design principle. NADINO integrates the DPDK-based F-stack [14] with the cluster-wide ingress for high-performance TCP/IP protocol processing and uses horizontal scaling to dynamically accommodate DPDK's polling cost.

The major contributions of NADINO[2] include:

- NADINO cleanly separates data-path and control-path tasks: RNIC handles fast-path packet I/O (*e.g.,* RDMA protocol processing, DMA), while DPU cores handle lightweight control path tasks such as operating RDMA QPs and multi-tenant scheduling. The DPU-based design outperforms the CPU-based design by 1.3~1.8× in RPS (§4.3).
- We show that having the serverless data plane built on top of two-sided RDMA yields up to 2.8× reduced latency and up to 2.7× improved throughput compared to variants using one-sided RDMA (§4.1.2).
- NADINO exploits the DPU in the serverless data plane to manage multi-tenancy in RDMA Fabrics. Evaluation in §4.2 shows that NADINO can fairly and precisely share RNIC bandwidth between tenants according to their weights.
- We introduce the first high-performance HTTP/TCP-to-RDMA ingress gateway designed for serverless environments. Compared to a traditional HTTP/TCP-based ingress gateway, even those using a high-performance TCP/IP stack, our design achieves a 3.4× reduction in end-to-end latency and a 3.2× increase in throughput.

Generality of the design. While NADINO is currently implemented on BlueField-2, the rationale for our design extends beyond their current hardware limitations. Even with new DPUs such as BlueField-3 that incorporate dedicated Datapath Accelerators (DPAs), a strict separation of concerns remains beneficial: the control plane is naturally lightweight and programmable, while the RNIC excels at line-rate data movement. In fact, NADINO can further benefit from the massive parallelism offered by DPAs (16 cores with 256 hardware threads [104]), which is well-suited for control plane tasks when handling large numbers of concurrent functions. NADINO's design principle of *offloading the data plane to hardware while retaining flexible control in software* remains valid across platforms.

[2]NADINO is available at https://github.com/ucr-serverless/NADINO.git

2 Background & Motivation

A Quick Primer on Serverless Data Planes. Fig. 1 shows the key components of a typical serverless data plane. Each serverless function comprises a *user function container* responsible for executing the application logic and a separate *sidecar container* [1, 4] for executing service mesh functionality, which helps to orchestrate the loosely coupled serverless functions. At the edge of this infrastructure is a *cluster-wide ingress gateway*, which serves as the entry point to the serverless cluster, facilitating tasks such as authentication. To facilitate invocations between serverless functions (when organized as function chains), serverless platforms generally rely on a *message broker* or *coordinator*. These data plane components typically communicate using *slow, kernel-based networking*, which incurs significant overheads, including data copies, context switches, interrupts, protocol processing (e.g., TCP/IP) and serialization/deserialization [26].

RDMA enables zero-copy networking to be distributed. RDMA facilitates *zero-copy*, *kernel-bypass* data movement between nodes equipped with RDMA-capable NICs. This capability extends zero-copy communication across multiple nodes in a serverless environment. It allows the widely used intra-node shared memory approaches to scale, without having to rely on the limitations of locality-aware placement strategies to maximize performance [27, 43, 78, 100]. Such placement strategies are often impractical due to node-wide resource constraints and the massive scale of production cloud applications [68]. Prior systems such as Junction [36], Pegasus [77], SURE [76], Demikernel [102], and mRPC [28] also pursue *kernel-bypass* communication across nodes, but all rely on software TCP/IP processing (*e.g.,* custom TCP stacks or F-stack). This software transport remains CPU-bound and incurs higher overhead (evaluated in §4.3). RDMA stands out by offloading transport-layer processing from the CPU to hardware.

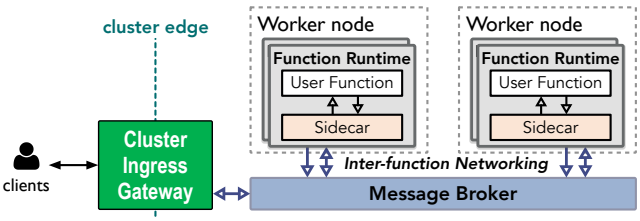

Figure 1. The architecture of serverless data plane (based on Knative's Serving/Eventing architecture [4]).

2.1 Challenges and Design Decisions

We identify RDMA as the ideal choice to enable a distributed zero-copy data plane for serverless computing. We focus on Reliable Connected (RC) RDMA transport in this work, ensuring in-order delivery and end-to-end reliability for application-layer transactions. This is also commonly used in practice [20, 32]. Despite early explorations [64, 67], several challenges remain to be addressed.

Challenge#1. Serverless computing lacks multi-tenancy support for RDMA Fabrics. Applications interact with the RNIC for data transfer via the QP abstraction [45], which requires privileged, low-level control of the RNIC. However, existing serverless platforms typically block low-level privileges from user code due to security concerns (*e.g.,* DoS attacks, possibly by malfunctioning clients that cause QP exhaustion [79, 98]). Further, direct access of QPs by serverless functions is also undesirable since it lacks the required isolation of shared RDMA resources across different tenants. Functions from different tenants would have to contend for locks to access the shared QP resources [32, 45, 59, 73]. This impacts performance due to lock contention as well as frequent cache line thrashing for QP buffers across CPU cores [32, 45]. Further, it is more difficult to enforce a fair allocation of QPs to tenants or prioritize traffic of important tenants. E.g., a 'rogue' tenant could occupy a set of QPs for a long time (i.e., possibly even deliberately not release the lock), thus starving other tenants of RDMA resources. Thus, there is no safe way for untrusted user code to exploit RDMA's high-performance data paths. Instead, RDMA must be offered as a managed network service, fully isolated from user-level code.

Design Implication#1: Exposing RDMA QPs directly to user functions is less feasible (and harmful) in a multi-tenant serverless environment. A managed network service (such as a node-wide network engine) helps enforce tenant-based isolation and manage QPs on behalf of user functions.

Challenge#2: Wimpy DPU Cores Slow Down On-path Processing when DPU Offloading. The DPU SoC can operate in the *on-path* mode or *off-path* mode [65, 95]. *On-path* mode allows full access to the in-flight data (since data is buffered in the DPU before being transferred further, as in Fig. 2 (**1**)). This enables the network engine on the DPU SoC to perform lightweight network processing, *e.g.,* for multi-tenancy support, before the data is relayed to the host functions. However, the on-path mode incurs additional data movement between the network engine on the DPU and the host. Even using the SoC DMA (which we find to be unfortunately very slow) can slow down the data path by up to **1.33**× as we assess in §4.1.1.

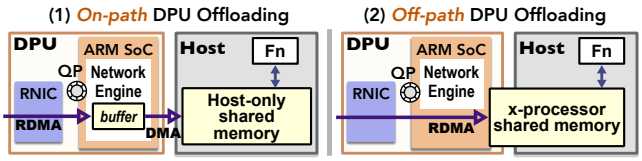

Figure 2. (1) On-path DPU offloading VS. (2) off-path DPU offloading (using cross-processor shared memory).

DPU SoCs configured in *off-path* mode are typically considered to have limited (or no) access to the packet data, as the data is moved directly between the integrated RNIC and host memory [65]. Thus, under the *off-path* mode, it is very difficult to have the equivalent network protocol processing on the DPU as the *on-path* mode. However, by creating a unified, cross-processor shared memory between the DPU and the host (CPU), we can expose the buffers of the host to the DPU *even in the off-path* mode (see Fig. 2 (**2**)). This is made possible by having a cross-processor memory map primitive on the DPU (*e.g.,* DOCA mmap library for the NVIDIA Bluefield DPU [15]). The integrated RNIC on the DPU performs RDMA operations to *directly* transfer data into the host shared memory. The *off-path* network engine running on the DPU manages the inter-node RDMA data paths by manipulating the RDMA QPs, without adding delays to the data path. The data is moved by the RNIC DMA (which runs at line rate) directly into the host memory, without encountering the limitation of the slow SoC DMA.

Design Implication#2: Cross-processor shared memory enhances *off-path* DPU offloading with full control of the data buffer while eliminating the additional data movement between DPU and the host (CPU) of the *on-path* mode.

Challenge#3: Selection of RDMA primitives for lock-free zero-copy communication. RDMA supports two modes of operation: one-sided and two-sided [45]. One-sided operations allow direct read or write access to buffers in the pre-registered memory region on a remote node without requiring the active involvement of the remote CPU. In contrast, two-sided operations adhere to traditional message-passing semantics, where both the sender and receiver actively participate in the communication. The sender delivers the data to a buffer explicitly posted by the receiver, offering greater flexibility in handling dynamic memory access requirements. There is no one-size-fits-all answer to whether one-sided or two-sided RDMA operation is better [24]. Conventional wisdom is that one-sided operations are faster than two-sided operations, while being highly CPU efficient [32].

The core of a lock-free zero-copy data plane in a distributed serverless environment lies in having a *unified memory pool* on each node. Because functions may execute asynchronously across nodes, simultaneous inter-node RDMA access and intra-node shared-memory accesses to a buffer in the *unified memory pool* can conflict, resulting in data races.

However, one-sided RDMA operations have a fatal flaw for our application scenario, which we define as the sender being "receiver-oblivious": The sender is not aware of whether the receiver's buffer is currently accessible for remote read or

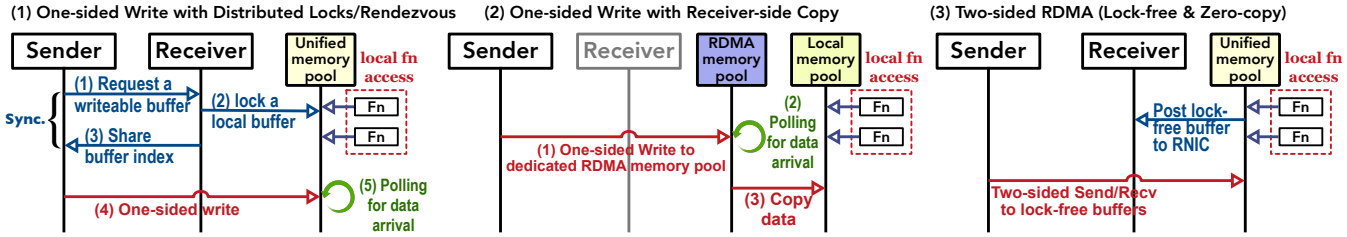

Figure 3. Selection of RDMA primitives for lock-free zero-copy serverless data plane.

write, *since a function on the receiver node may already be using it*, also leading to potential data races.

Two potential workarounds exist: (1) employ distributed locks [21, 38, 99] or Rendezvous-based synchronization [88] to coordinate remote one-sided RDMA read/write operations with local shared memory processing (Fig. 3 (1)). However, this incurs additional synchronization overhead across nodes. or (2) dedicating an RDMA-only memory pool on the receiver node, exclusively used for remote one-sided operations (Fig. 3 (2)), which avoids data races by isolating it from the memory pool shared by local functions. However, this introduces an additional data copy, which undermines the goal of having an efficient zero-copy design.

Why two-sided RDMA is a better fit. Two-sided RDMA operations inherently avoid the sender being "receiver-oblivious" (see Fig. 3 (3)). The receiver explicitly posts a buffer for incoming data, ensuring an exclusive ownership transfer that eliminates conflicts from multiple writers (either local or remote). This built-in coordination removes the need for distributed locks or dedicated RDMA memory pools that incur copies, while avoiding data race conditions entirely. Our assessment in §4.1.2 shows that two-sided RDMA is **2×-2.8×** faster than one-sided write using distributed locks/Rendezvous, and up to **1.6×** faster than one-sided write with receiver-side copy, which validates our choice. The additional CPU overhead of two-sided RDMA (primarily from the receiver side) is often a concern [24, 32, 38]. However, by delegating the two-sided RDMA operations to the DPU, we can preserve CPU cycles, while also maintaining the advantages of two-sided RDMA.

Design Implication#3: Two-sided RDMA is fundamentally more compatible with the requirements of a distributed, lock-free, zero-copy data plane in serverless environments, compared to one-sided RDMA.

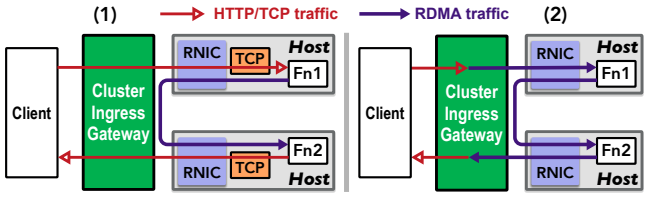

Figure 4. Alternatives of transport protocol adaptation: (1) "Deferred" transport conversion within cluster; (2) Early transport conversion at cluster ingress (used by NADINO).

Challenge#4: Transport Protocol Incompatibility. Fig. 4 (1) illustrates the transport mismatch described in §1: the existing RDMA-based serverless data plane relies on a TCP/IP stack on each worker node to terminate HTTP/TCP traffic from external clients. This "deferred" transport conversion results in duplicate HTTP/TCP/IP processing even within the serverless cluster. Our evaluation in §4.1.3 reveals significant drawbacks in this approach. Specifically, the HTTP/TCP-based cluster ingress exhibits poor scalability, increasing the end-to-end latency by up to **11.7×** and reducing the RPS by **11.4×**. An effective way to mitigate the redundant TCP/IP processing cost within the serverless cluster is to terminate the external client-side transport protocol at the *earliest* point possible *i.e.,* at the cloud cluster edge, as shown in Fig. 4 (2).

Design Implication#4: Dedicating transport protocol adaptation to the cluster-wide ingress at the edge eliminates redundant TCP/IP processing within the cluster—an often-overlooked optimization that warrants more attention.

2.2 Related Work

Optimizing serverless data plane: SPRIGHT [78], NightCore [42], and many others [29, 84, 100] rely only on local shared memory processing, which is not scalable. While RMMap [67] and FUYAO [64] utilizes one-sided RDMA to construct their data plane, they are subject to the limitation of one-sided RDMA (§2.1). Besides, none of them support multi-tenancy for RDMA and are limited by the traditional HTTP/TCP cluster ingress. As summarized in Table-1, NADINO goes beyond these prior designs with a comprehensively optimized data plane. Other optimizations include lightweight sidecar implementations to reduce service mesh overhead [76, 78, 82, 87], and direct inter-function invocation [29, 64, 78, 101] to bypass the intermediate coordinator, thus completely avoiding the substantial cost imposed by such middleware. These efforts complement NADINO.

Table 1. Comparison of Existing High-Performance Serverless Dataplane Systems

Systems	Multi-tenancy Support	Distributed Zero-copy	DPU Offloading	Eliminate proto. processing within cluster
NightCore [42]	✗	✗	✗	✗
SPRIGHT [78]	✗	✗	✗	✗
FUYAO [64]	✗	✗	✓	✗
RMMAP [67]	✗	✓	✗	✗
NADINO	✓	✓	✓	✓

RDMA: Several complementary efforts seek to improve various aspects of RDMA, including scalability [59, 71, 73, 93], and security [98]. Performance isolation of RDMA has been a key focus in multi-tenant environments [52, 53, 66, 103, 104]. Solutions like [91] and [96] provide in-kernel mechanisms to isolate RDMA QPs from untrusted applications but require kernel changes. NADINO enables the RDMA isolation through a userspace software solution (DNE), leveraging DPU offloading to eliminate CPU cost while also providing physical protection against CPU-based attacks. Recent work done concurrently with NADINO, SCR [104], also proposed a userspace software solution to enable RDMA isolation. It is built on the latest Nvidia BlueField-3 DPUs, allowing them to benefit from BlueField-3's Datapath Accelerator for its high parallelism and hardware-offloaded execution.

DPU/SmartNIC offloading has been widely explored to enhance various aspects of cloud data centers, including storage systems [69, 70, 86, 105], distributed file systems [49], multi-tenancy [47, 62], request scheduling and load balancing [31, 57, 61], TCP offloading [50, 74, 83], service mesh acceleration [58], distributed transactions [81], ML training/serving [54, 97]. [106] provides isolation for SmartNIC-offloaded network functions from different tenants. Specific to serverless computing, [30] offloads serverless functions to P4-enabled Netronome SmartNICs [5] but is constrained by a limited programming model. [64] uses DPU to offload the coordinator (Fig. 1), reducing CPU costs. NADINO goes beyond [64] by unleashing the potential of DPU offloading through cross-processor shared memory.

3 Design of NADINO

3.1 Overview

Fig. 5 shows the core components in NADINO: **(1)** Each worker node in NADINO has an associated DPU[3], which runs a lightweight DPU Network Engine (DNE) (§3.2). By granting the DNE *exclusive* access to RDMA QPs, NADINO enforces strict per-tenant network isolation at the RNIC (§3.3). **(2)** NADINO treats each function chain as an independent 'tenant', with each tenant owning a unified memory pool (physically located on the host) for exclusive access. This enables per-tenant memory isolation (§3.4.1). By extending the unified memory pool into a cross-processor shared memory (§3.4.2) between the DPU and the CPU (host), the DNE stitches the inter-node (RDMA) data plane into intra-node (shared memory) data plane, while being off-path. NADINO employs the pool-based buffer management for fast buffer allocation and reuse. **(3)** To facilitate lock-free communication in NADINO's data plane, we employ two-sided RDMA for inter-node data transfers and use token passing for intra-node data transfers (§3.5.1). On the host, we use the eBPF's

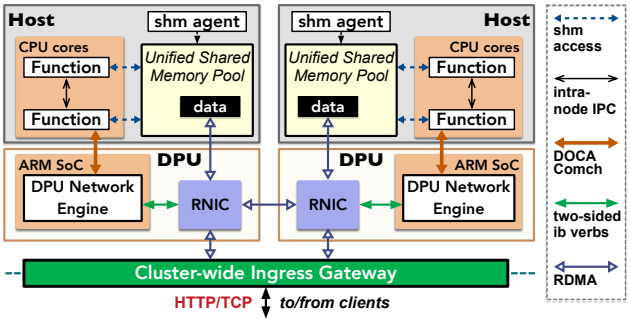

Figure 5. An overview of NADINO. DNE, as a trusted service, is physically isolated on the DPU. It is not co-located with untrusted user functions that run on the CPU.

SK_MSG IPC from [78] to hand off a buffer descriptor between co-located functions, taking advantage of its event-driven execution model and sidecar extensibility. We rely on NVIDIA's DOCA Communication Channel (Comch [10]) for cross-processor exchange of buffer descriptors between the DNE and host functions, which was the most functionally capable option on BlueField DPUs, as we evaluate in §3.5.4. **(4)** The ingress gateway is deployed on independent server nodes at the cluster edge, rather than on nodes acting as workers for serverless functions. HTTP/TCP connections are terminated at the cluster-wide ingress gateway and the payload data is then transferred to nodes within the cluster over RDMA Fabrics, and vice versa (§3.6).

Threat model and trust model in NADINO. NADINO's threat model considers three threat sources for the network engine: (1) CPU-related microarchitectural attacks (*e.g.,* Meltdown [63], Spectre [51]). (2) CPU interference from user functions via shared core contention. (3) RDMA interference between user functions (DoS attacks via QP exhaustion [79, 98]). Both (1) and (2) are more difficult to control when the network engine shares CPU cores with untrusted functions. (3) is a typical example of security concerns of user code access to low-level control of the RNIC. Direct exposure of QPs to malicious functions can result in the functions intentionally creating many RC QPs to trigger frequent cache eviction on the RNIC, thus impacting the RDMA performance for others.

To address threats (1) & (2), the DNE is physically isolated on the DPU, and never co-located with untrusted user functions on the CPU.[4] NADINO addresses threat (3) by using the DNE (together with DOCA comch) to proxy the access to RDMA QPs on behalf of the user functions.

NADINO assumes an asymmetric trust model: Users trust the NADINO infrastructure (*i.e.,* DNE). However, NADINO does not trust user functions that may be buggy or malicious. The DNE runs as a privileged service on the DPU and forms part of the TCB, which includes DPU's hardware (ARM cores, DMA engines, RNIC cores) and software (DOCA runtime,

[3]Our implementation leverages the NVIDIA BlueField-2 [6], but our design is generally applicable to other BlueField models.

[4]We do not allow user functions to be offloaded to DPU, *i.e.,* no user access to DPU in NADINO.

OS). To restrict access to shared data, only functions from the same tenant that trust each other can use shared memory processing (see §3.4). An explicit CPU-based data copy is used for communication across tenants. NADINO also utilizes the sidecar to enforce the necessary access control when exchanging data between functions. To reduce the sidecar overhead, NADINO uses a streamlined eBPF sidecar [3, 78] and also a node-wide shared "sidecar" [28, 87] (as part of the DNE) to replace the container-based, individual sidecar.

3.2 Off-path DPU Network Engine (DNE)

NADINO's DNE consists of a *core thread* and multiple *worker threads*. The core thread manages memory mapping from the host, registers memory regions to RNIC (§3.4.2), and establishes DOCA Comch channels with host functions. The worker thread handles the data packet processing within a non-blocking, *run-to-completion* event loop, processing each packet through all the transfer stages without interruption. This *run-to-completion* design eliminates overheads (*e.g.,* scheduling, context switch) that may tax the wimpy DPU cores, and further improves the cache locality [22, 102].

Run-to-completion Event Loop: Fig. 8 describes the *run-to-completion* event loop, including both transmit (TX) and receive (RX) stages. In the TX stage, the DNE consumes a buffer descriptor from the source function, determines the destination node via the inter-node routing table, and selects the least-congested RC connection. It then wraps the descriptor into an RDMA work request (WR) and posts it to the RNIC for transmission. Upon data arrival, the RX stage polls the completion queue entries (CQEs) and retrieves the corresponding buffer descriptors via a receive buffer registry table lookup (more details in §3.5). It then extracts the destination function ID, and forwards the descriptors to the function using the corresponding Comch endpoint.

3.2.1 Cost of NADINO's isolation. To quantify the cost of DNE, we use a pair of echo client/server functions placed on two worker nodes. Each function runs on the host with one dedicated CPU core. The DNE runs in off-path mode. We compare the DNE setup with two native RDMA baselines, where the functions directly use two-sided RDMA send/receive with a single RC QP: (1) *native RDMA (CPU)*, in which the functions run on host's CPU; and (2) *native RDMA (DPU)*, in which the functions run on the DPU. The latter baseline allows us to quantify the inherent penalty of executing RDMA primitives on the wimpy DPU cores.

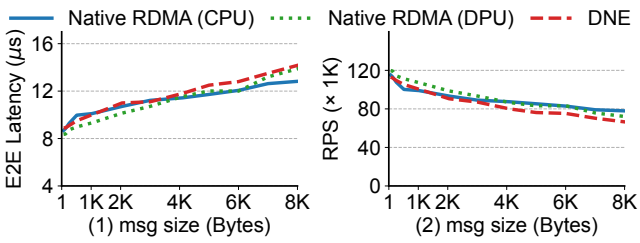

Figure 6. Isolation cost of NADINO's DNE: (1) mean end-to-end latencies; (2) RPS. All settings use two-sided RDMA.

Fig. 6 shows that the cost introduced by DNE as an additional isolation layer is limited. The performance of DNE aligns closely with native RDMA (DPU) and native RDMA (CPU). This validates the effectiveness of the run-to-completion event loop and off-path design. The performance overhead incurred by executing RDMA primitives directly on the wimpy DPU cores is minimal.

3.3 Multi-tenancy support in NADINO's data plane

NADINO leverages the DNE to ensure fair sharing of RNIC bandwidth among co-located tenants. Traffic from tenants of greater importance is prioritized using a Deficit Weighted Round Robin-like [85] scheduler. By leveraging tenant-specific weights, the DNE prioritizes functions accordingly, enabling more frequent transfers for higher-priority tenants compared to lower-priority ones.

RC connection management: NADINO uses RCQPs[5] to establish dedicated, point-to-point reliable connections between peer nodes. Each tenant is allowed to utilize multiple RCQPs (proxied by DNE), where each comprises a Send Queue (SQ) and a Receive Queue (RQ). To reduce the QP memory footprint, all of a tenant's RCQPs share a single RQ, which is posted with buffers from that tenant's private memory pool. This guarantees the RNIC delivers incoming data into the correct pool. All RCQPs on a given node share a single Completion Queue (CQ).

Establishing RC connections between peer nodes incurs significant overhead, as the connection setup time is non-negligible (of the order of tens of milliseconds) [59, 96]. To mitigate this, we employ a node-specific management approach, where a pool of established connections (to a remote peer node) is managed by the DNE of this node. In order to maintain a large number of established RC connections with negligible overhead, we employ the "shadow" QP mechanism from [55] to categorize RCQPs in the pool into *active* and *inactive* QPs: A RCQP is considered active when it has WRs queued; otherwise, it is inactive [94]. Inactive RCQPs consume *no* RNIC resources [55, 59]. As such, we only need to limit the total number of active RCQPs per node to avoid cache thrashing on the RNIC. The DNE dynamically activates or deactivates RC connections in proportion to the load between node pairs without any cross-node QP state synchronization [55].

3.4 Memory Subsystem in NADINO

Fig. 7 shows the architecture of memory subsystem in NADINO. The unified shared memory pool includes a set of buffers created on the host. We use hugepage memory (2MB size each) to create buffers, which helps reduce the memory footprint of

[5]We use "RC-based connection" and "RCQP" interchangeably.

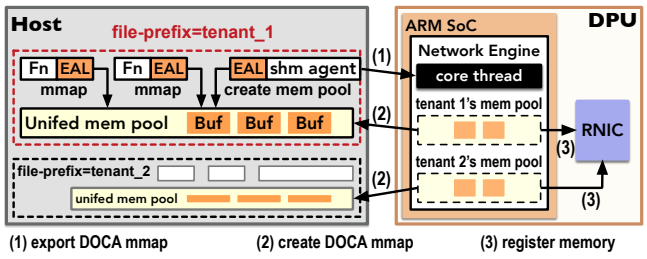

Figure 7. The memory sharing and isolation design in NADINO (a single node view). The memory pool is created on the host main memory and mapped to DPU.

the Memory Translation Table on the RNIC cache [93]. The unified memory pool is mapped to the DPU and registered as memory regions for the RNIC by DNE.

Pool-based buffer allocation and recycling: NADINO utilizes the memory pool to reserve a number of equal-size buffers before they are actually used by a function. When the function needs a new buffer, it gets one from the memory pool instead of calling a malloc-like API (e.g., glibc) to dynamically allocate memory each time it is needed. This helps reduce allocation latency, fragmentation, and prevent memory leaks [39, 46, 107]. We currently only support a fixed-sized memory pool. We use the DPDK memory allocator interface to allocate a new buffer (`rte_mempool_get()`) and to release an existing buffer to the pool (`rte_mempool_put()`).

3.4.1 Memory isolation across tenants.

NADINO uses the DPDK memory allocator to take advantage of DPDK's file-prefix feature [13] to enforce per-tenant memory isolation, and generate memory-mapped files specific to the memory pool of a tenant (*i.e.,* a function chain). The memory-mapped files contain the configuration (*e.g.,* the virtual addresses of the hugepages in use [13]) of the memory pool created. Each tanent has a distinct file-prefix, bound to a memory pool for exclusive access (as "tenant_1" versus "tenant_2" in Fig. 7).

We create a per-tenant shared memory agent as the DPDK primary process, to set up the memory pool prior to the function startup (Note that the shared memory agent is not involved in the data transfer). Functions (as DPDK secondary processes [13]) utilize corresponding file-prefix to load the generated memory-mapped files and obtain the memory configuration. This enables functions to map to the shared memory pool through DPDK's Environment Abstraction Layer (EAL [12]) and facilitates isolation between the memory pools of different tenants.

3.4.2 Cross-processor shared memory.

NADINO achieves cross-processor shared memory by mapping the unified memory pool (physically located in the host's main memory) to the DPU. The cross-processor shared memory mechanism is implemented by using the mmap APIs from NVIDIA's DPU programming library (called DOCA [15]).

As shown in Fig. 7, the DNE works in tandem with the shared memory agent to map the unified shared memory pool from the host: (1) The shared memory agent on the host generates an export descriptor (DOCA's mmap descriptor output parameter which is used to represent memory ranges in remote system memory space) of the local memory pool. This is done using `doca_mmap_export_pci()` (to grant DPU ARM core access) and `doca_mmap_export_rdma()` (to grant RNIC access). The shared memory agent transmits it to the DNE via the DOCA Comch [10]. (2) Upon receiving the export descriptor, the DNE establishes a remote memory map (using `doca_mmap_create_from_export()`). (3) With this setup complete, the DNE possesses all the necessary memory from the host, thus enabling the registration of the memory to the RNIC.

3.5 Intra-node & Inter-node data transfer

Since NADINO's data plane spans both intra-node shared memory and inter-node RDMA, each uses distinct communication primitives. To hide the intricacies of different data transfers (shared memory or RDMA) from the user code, we introduce a unified I/O library built-into NADINO's function runtime. The I/O library offers generic point-to-point messaging interfaces (`send()`, `recv()`) for operating on NADINO's data plane, sparing developers from selecting the correct transport. Beyond simple messaging, the API is extensible: we layer RPC semantics and DAG-style dataflows on top of the same primitives, as demonstrated in Online Boutique application (evaluated in §4.3).

The I/O library, once invoked by the user code, transparently determines the intra-/inter-node data path through the routing phases as shown in Fig. 8: If the intra-node routing query confirms that the destination function resides on the same node, the I/O library dispatches data using the intra-node shared memory IPC to exchange descriptors between functions (green arrow in Fig. 8). Otherwise, it hands off the data to the DNE, which handles inter-node RDMA forwarding (violet arrows in Fig. 8).

3.5.1 Lock-free buffer ownership transfer.

The use of two-sided RDMA prevents concurrent writes to the receiver-side buffer during inter-node transfers. For intra-node communication, NADINO employs explicit token-passing to transfer buffer ownership between local functions (or between host functions and DNE), which emulates the behavior of single-producer single-consumer ring that guarantees lock-free buffer access. NADINO implements token-passing using semaphores: consider a linear chain of three functions ($A \rightarrow B \rightarrow C$), where each pair of communicating functions shares a common semaphore. Each semaphore is initialized to 0. When A completes its task, it calls `sem_post` on B's semaphore (set it to 1), handing off buffer ownership to B. B waits on the semaphore (blocked on `sem_wait`) before

gaining ownership of the buffer. After processing, *B* similarly calls `sem_post` on *C*'s semaphore to transfer ownership further downstream. This ensures the ordered transfer of access from an upstream producer to a downstream consumer, following the call graph of the function chain.

NADINO's buffer lifecycle management adheres to exclusive ownership semantics, *i.e.,* only the buffer owner (function or DNE) can read, write, or recycle the buffer, which is suitable for working with the message-passing communication model using two-sided RDMA. This helps ensure that buffers are not inadvertently modified (data corruption) or prematurely released (may cause segmentation faults).

3.5.2 Inter-node RDMA transfers. Our two-sided RDMA design has the receiver-side DNE post a buffer (included in the WR) to the RNIC for receiving data from the sender. To track the buffer where the data is RDMA'ed into, we maintain a receive buffer registry (RBR) table on the DNE to map the WR to the posted receive buffer (similar functionality is already integrated when using DOCA APIs). The DNE core thread asynchronously monitors the number of CQE consumed by different tenants (via shared counters updated by the RX stage) and posts an equal amount of receive buffers to the RQ of corresponding tenant. This ensures the receiving RNIC always has enough buffers to receive incoming data for corresponding tenants (shown as red arrows in Fig. 8).

3.5.3 Intra-node shared memory IPC. The zero-copy shared memory processing between functions depends on delivering the descriptor of the shared memory buffer. The ownership of the descriptor determines the access to the shared memory buffer, using the data pointer contained in the descriptor. NADINO utilizes eBPF's `SK_MSG` [2] for intra-node IPC (same as [78]). The transmission of `SK_MSG` between sockets entirely bypasses the kernel protocol stack, thus eliminating unnecessary processing, and making it a desirable capability to transfer the small descriptors.

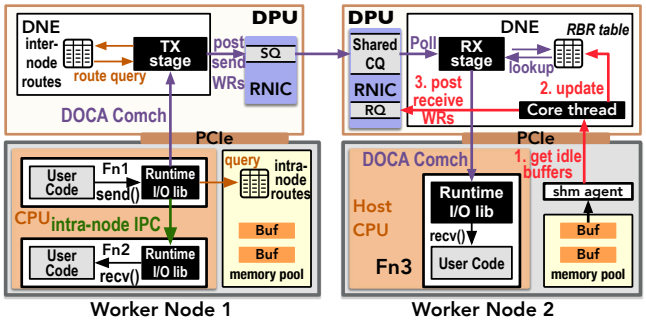

Figure 8. *Lock-free, zero-copy* data transfer in NADINO. Green arrow depicts intra-node shared memory data transfer; Violet arrows depict inter-node data transfer using two-sided RDMA; Red arrows depict RBR table updates (only shown on Node 2).

3.5.4 Cross-processor communication channel. We deploy the DNE as the single Comch server instance to communicate with functions (which are Comch clients) on the host. The DNE busy-polls all monitored function endpoints within its event loop for buffer descriptors. We use epoll to enable event-triggered reception by the function, ensuring a more efficient and practical approach that retains good function density in a serverless environment.

DOCA Comch offers two communication channel variants [10], which can be used for exchanging 16B buffer descriptors between NADINO's DNE and host functions: (1) Comch-E uses event-driven send/receive primitives on top of blocking Linux epoll, while (2) Comch-P uses a producer–consumer ring with busy polling (lowest latency but ties up a core per function). We compare both against a TCP baseline (we avoid intra-node RDMA as it is insecure, allowing user functions direct access to QPs violating multi-tenancy, as discussed in §2.1. In contrast, Comch allows the DNE to disconnect misbehaving tenants). In our test, multiple functions on the host issue back-to-back descriptor sends to a single-core DNE on the DPU and await replies.

Fig. 8 shows the comparison results: TCP suffers the highest latency due to kernel and protocol overhead. Comch-P cuts latency by >8× versus TCP but hits scalability limits (one core per function) and overloads beyond 6 functions. Based on our examination, it is caused by the use of an internal epoll interface by `Comch-P` for its "busy" polling (DOCA implements its event loop called "Progress Engine" [11], which is actually performed via non-blocking `epoll_wait` and introduces kernel-related overheads, which hopefully will be improved upon in the future in this closed-source implementation). Comch-E, while is slower than Comch-P, still outperforms TCP by 2.7× –3.8× and delivers stable. Its event-driven design without dedicated cores makes it the most practical choice for a dense, multi-tenant serverless platform. We choose to use Comch-E in NADINO.

3.6 NADINO's Cluster-wide Ingress Gateway

The ingress gateway employs a master-worker model, as shown in Fig. 10. The master process manages control-plane tasks, including loading the configuration and horizontal

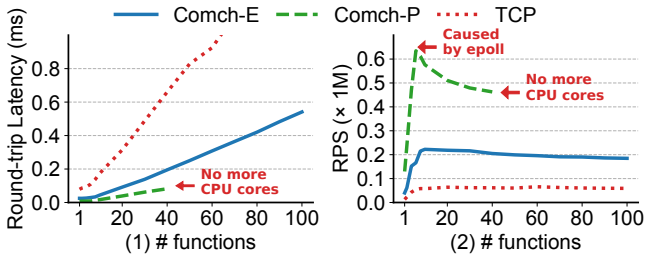

Figure 9. Evaluation of viable communication channels between DPU and host (CPU): (1) Round-trip latency (in milliseconds); (2) Buffer descriptor transfer rate (RPS).

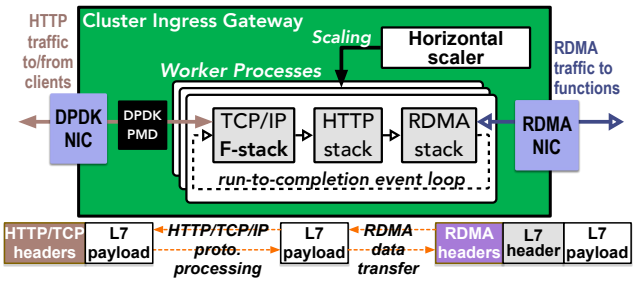

Figure 10. HTTP/TCP-to-RDMA transport protocol adaptation at the cluster-wide ingress gateway.

scaling of the gateway worker processes. Worker processes handle all data-plane tasks, including TCP/IP protocol processing, HTTP processing, and RDMA transmission. Each worker process performs all data-plane tasks for transport protocol adaptation, using a run-to-completion event loop (busy polling loop), just like the DNE (§3.2). We enable batching in the event loop to improve concurrency. We base the implementation of the ingress on NGINX [34] (v1.25.2) to leverage its full-fledged HTTP processing implementation. We integrate the DPDK-based F-stack [14] in our ingress's busy polling loop taking advantage of its high-performance TCP/IP protocol processing in the userspace instead of the slower kernel protocol stack. Deploying NADINO's cluster gateway does not require a DPU. A node equipped with a regular RNIC and a DPDK-compatible NIC is sufficient.

Event-driven architecture: Callback functions are registered in the event loop to handle HTTP protocol processing (as in NGINX [34]) and RDMA send/receive events. When an event occurs (*e.g.,* on a new connection or data is available to read), the corresponding handler is called to process it. We leverage Receive Side Scaling (RSS [26]) to distribute traffic from external clients evenly to different worker processes (pinned to specific CPU cores), thus achieving the equivalent effect of Accelerated Receive Flow Steering (aRFS) without additional NIC hardware support [26].

Horizontal scaling: We allocate each worker process a dedicated CPU core to execute the busy polling loop. To optimize resource usage during low demand periods and ensure performance during peak traffic, we horizontally scale the number of worker processes based on the load level of the worker processes using a simple but effective hysteresis policy. We refine the CPU time measurement in the worker process event loop to collect the aggregated CPU time spent on data-plane tasks. This enables us to quantify CPU utilization dedicated to the "useful" work performed by the worker process. Once the average CPU utilization across existing worker processes reaches 60%, the master process spawns a new worker process to handle the increased load. Conversely, when the average CPU utilization across existing worker processes drops below 30%, the master process terminates a worker process to conserve CPU resources.

3.7 Other Considerations

Cold-start mitigation: NADINO focuses on optimizing the serverless data plane for function chains. Although it does not directly address cold-start delay, NADINO leverages SPRIGHT's keep-warm policy to mitigate cold-start impact. NADINO also complements prior efforts such as Catalyzer [33] and FaasCache [37]. Furthermore, as discussed in §3.3, NADINO amortizes QP churn through connection pooling across DNEs, avoiding the setup overhead for short-lived connections (similar to [55, 73]).

SR-IOV VF-based isolation: Although SR-IOV provides hardware-level isolation by exposing separate Virtual Functions (VFs) through the PCIe, it does not enforce strict isolation between user functions and the RNIC itself, as NADINO's DNE does. Particularly, even if each tenant is assigned a distinct VF, these VFs still contend for the same shared RNIC resources such as caches. As demonstrated in Harmonic [66], a malicious tenant can exploit this shared microarchitectural state via the VFs, thereby potentially exhausting the RNIC's cache (*e.g.,* MTT/MPT entries), causing interference with other tenants. Thus, SR-IOV alone is insufficient to guarantee multi-tenant isolation in RDMA environments. A DNE-like software layer remains essential to mediate access to the RNIC and enforce strong isolation across tenants.

Mitigating vendor lock-in: We acknowledge that NADINO currently depends on DOCA APIs (*e.g.,* Comch and mmap), tying it to NVIDIA's BlueField. However, this reliance reflects an engineering decision rather than a fundamental limitation. Our design is portable in principle to other DPU or SmartNIC architectures. For example, AMD's Alveo SmartNIC supports similar capabilities: it enables direct host memory access from the SmartNIC,[6] similar to DOCA's mmap. It also offers a host-SmartNIC communication channel via PCIe BAR-mapped memory[7] (*i.e.,* a ring buffer shared between host and SmartNIC), akin to DOCA's comch. Other SmartNICs, such as Netronome and Intel's IPU, also provide PCIe BAR access that can support similar abstractions. Adapting NADINO to these platforms would primarily involve engineering work to use a vendor's hardware interfaces, without fundamental changes to the design.

4 Evaluation

We first perform a set of microbenchmarks to showcase the improvement by the various design choices made in NADINO. We then show the benefit of NADINO with a system-level evaluation using real serverless workloads.

Testbed: Our testbed consists of four server nodes. Each node is equipped with two 40-core Intel Xeon Gold 6148 CPUs (2.4GHz base, 3.7GHz max) and 500GB of RAM, with hyper-threading and NUMA enabled. We run Ubuntu 22.04

[6] https://xilinx.github.io/XRT/master/html/hm.html
[7] https://xilinx.github.io/AVED/amd_v80_gen5x8_exdes_1_20231204/AVED+-+Host+to+Card+Communication.html

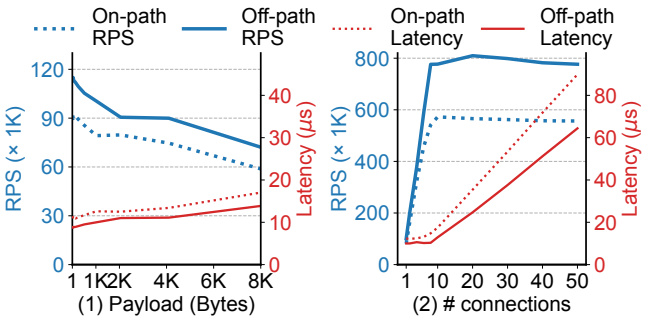

Figure 11. NADINO's off-path DNE (enhanced with cross-processor shared memory) VS. on-path DNE: (1) RPS with varying payload sizes (single connection); (2) RPS under different concurrency levels (1KB payload size).

with kernel 5.15 on each node. Two of the nodes are used as workers for deploying user functions. Each worker node has an NVIDIA BlueField-2 DPU (with an integrated ConnectX-6 RNIC) to act as the DNE. The DPU is installed with DOCA SDK 2.9.0. We use another node (called the ingress node) to deploy the cluster-wide ingress gateway and the remaining node without the DPU as the client node to generate the workload from external clients. The ingress node has two separate ConnectX-6 RNICs: one used for RDMA communication with user functions and the other to function as the Ethernet NIC to communicate with the client node. The DPUs on the worker nodes and the ConnectX-6 RNIC on the ingress node are interconnected by a 200 Gbps switch. The Ethernet NIC on the ingress node and the client node are connected through another distinct 200 Gbps switch.

4.1 Microbenchmark Analysis

4.1.1 Cross-processor shared memory. We compare the performance difference between the offloading model with off-path DPU processing in NADINO (enhanced with cross-processor shared memory) versus offloading with on-path DPU processing. The two options are depicted in Fig. 2. In the on-path mode, the DNE buffers data locally (SoC memory) when moving the data between a function on the host and the RNIC. The on-path DNE uses IB verbs to interact with the RNIC, while using the DMA engine on DPU SoC to move data between local buffers and the shared memory pool on the host. Note that we do not use intra-node RDMA to move data between DPU and the host as it adds 2 PCIe transfers [95]. On the other hand, RNIC DMA avoids this added overhead. It also avoids interference with inter-node RDMA traffic [28]. We use a echo server/client function pair to measure the round-trip time and throughput. The server and client functions are deployed on different nodes.

Fig. 11 shows the RPS and latency comparison: The off-path DNE achieves up to 30% RPS improvement with more than 20% latency reduction compared to the on-path mode under various payload sizes and concurrency levels. The

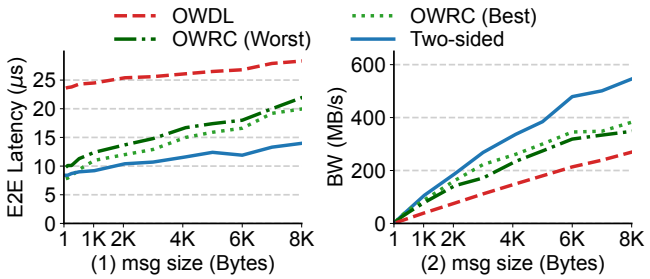

Figure 12. Performance impact of selecting RDMA primitives: (1) mean end-to-end latencies; (2) RPS. NADINO uses two-sided RDMA.

shortcoming of on-path mode is manifested under high concurrency traffic due to the poor processing capability of the DPU DMA engine [95]. This shows the necessity of using cross-processor shared memory between the host and DPU, thereby eliminating the need for additional data copies (and using DMA instead). At low concurrency, the RPS of on-path mode is close to off-path mode because the DPU's DMA engine is not overloaded. At this light load, the on-path mode can fully exploit the low-latency data copy of the DPU's DMA engine (only 2.6μs for 64B DMA read [95]). But as the number of connections increases, the on-path mode's latency grows much more quickly compared to the off-path mode.

4.1.2 Selection of RDMA primitives. To validate our choice of two-sided RDMA, we benchmark three alternatives in Fig. 3. We configure two DNEs on different worker nodes to act as a pair of echo client/server and allocated one core to each. Since the completion of the one-sided RDMA write is not visible to the receiver, it has to poll for the data arrival (following FARM's receiver-side design [32]). The receiver then echos the data. Note that repetitive measurements with OWRC (one-sided write w/ receiver-side copy) can introduce bias on the echo server. Specifically, the source buffer in the RDMA memory pool and the destination buffer in the local memory pool may both be cached. This can lead to optimal cache locality and an artificially improved copy performance. This scenario is unlikely to hold in real-world workloads. To account for this, we introduce a variant of OWRC that enforces main memory access during data copies by flushing the TLB, ensuring a worst-case evaluation. We refer to the OWRC with artificial cache locality as `OWRC-Best` and the one with enforced main memory access as `OWRC-Worst`.

Fig. 12 (1) shows the latency differences between the various approaches. A single one-sided write can take as little as 4μs, but exchanging locks and copies introduce non-negligible overhead. Two-sided RDMA achieves the lowest latency (8.4μs for 64B messages) by eliminating data races in the data plane (§2.1), thereby streamlining data transfers over RDMA. With a 4KB message size, two-sided RDMA (11.6μs) reduces latency by 1.3×, 1.5× and 2.3× compared to `OWRC-Best` (15μs), `OWRC-Worst` (16.7μs) and `OWDL` (one-sided write w/ distributed locks, 26.1μs), respectively. Looking at

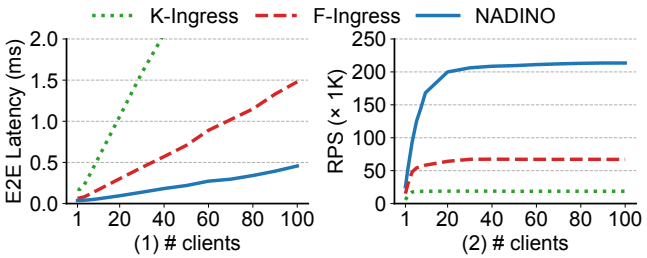

Figure 13. Performance of various cluster ingress designs: (1) mean end-to-end latencies; (2) RPS with varied # of clients.

throughput in Fig. 12 (2), NADINO is up to 1.3× and 1.4× higher than `OWRC-Best` and `OWRC-Worst`, and is more than 2.1× more than `OWDL`. These demonstrate the advantages of adopting two-sided RDMA in NADINO.

4.1.3 Transport protocol adaptation. We evaluate end-to-end latency savings and improved scalability from consolidating the transport protocol adaptation to NADINO's cluster edge (Fig. 4 (2)). As baselines, we compare two NGINX-based ingresses The baselines we compared with are two NGINX-based variants of Fig. 4 (1): one uses the interrupt-driven Linux kernel TCP/IP stack (denoted as *K-Ingress*); the other uses the more performant DPDK-based F-stack [14] for protocol processing tasks (denoted as *F-Ingress*). We deploy an HTTP server function on the worker node to echo the requests from "external" clients (wrk [18]), relayed by the cluster ingress. For the variants of Fig. 4 (1), we use F-stack on the worker node for TCP/IP processing.

We assign one CPU core to the cluster ingress in this experiment. The cluster ingress is likely to have to consistently handle a high volume of requests across concurrent connections. As shown in Fig. 13 (1), the end-to-end latency of NADINO's ingress is significantly lower than *K-Ingress* and *F-Ingress*, since NADINO replaces the slow software-based transport processing with RDMA for intra-cluster networking. NADINO also increases RPS by up to 11.4× and 3.2× RPS compared to *K-Ingress* and *F-Ingress*, respectively. Since the variants of Fig. 4 (1) terminate the TCP at the worker node, this in fact doubles TCP/IP processing work at the cluster ingress, thus affecting scalability.

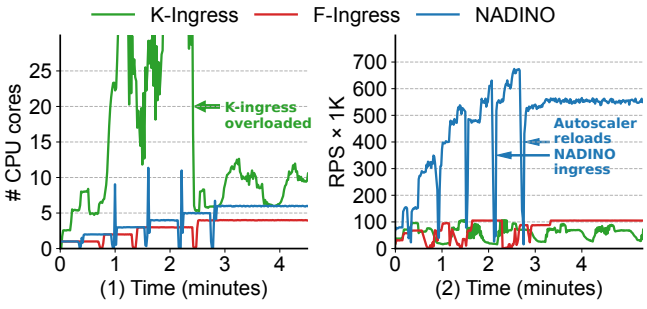

Figure 14. Effect of horizontal scaling of NADINO's ingress: (1) CPU usage time series of the cluster ingress; (2) RPS times series.

To assess the horizontal scaling of NADINO's ingress, we increase the load by adding a client every 10 seconds, and each client is configured to fully use up a CPU core to generate the highest load it can (with multiple connections). We also adapt our autoscaler to support the *F-Ingress* for a fair comparison. Fig. 14 (1) shows the CPU usage over time. The horizontal scaling in NADINO's ingress alleviates the busy-polling overhead by adjusting the number of active worker processes to match load, while retaining the low-latency benefit of busy polling (see §3.6). NADINO's ingress uses much less CPU while achieving more than 5× RPS compared to interrupted-driven *K-Ingress*. The *K-Ingress* is quickly overloaded after using up all CPU cores (at around the 2.5 minute mark) and results in most of the clients becoming disconnected due to the lack of a response. Note that the scaling procedure in NADINO's ingress triggers a brief service interruption due to the restart of the worker processes (see Fig. 14 (2)). This can be avoided by enabling load balancing across multiple NADINO ingress instances.

4.2 Multi-tenancy Support for RDMA

Tenant types and workload: We evaluate NADINO's multi-tenancy support (implemented as a Deficit Weighted Round Robin policy in DNE) for fair inter-node bandwidth sharing. We compare NADINO with a first-come-first-served ("FCFS") DNE that lacks explicit multi-tenancy handling. In this experiment, we configure the DNE to sustain a maximum throughput of approximately 110K RPS when it fully utilizes the assigned single DPU core. We consider three tenants that compete for the bandwidth, each assigned a different weight: Tenant-1's weight is 6; Tenant-2's weight is 1; Tenant-3's weight is 2. Every tenant runs a pair of client/server functions, placed across two worker nodes to trigger inter-node data transfers via the DNE. We induce contention by making Tenant-2 and Tenant-3 generate periodic surges (Tenant-3 is slightly more bursty). Tenant-1 (green line) remains active for the entire 4-minute experiment, while Tenant-2 (dotted line) joins at 20s and exits at 3m20s. Tenant-3 runs between 1m30s and 2m30s (blue line).

Observations: Fig. 15 shows the bandwidth distribution among tenants, using (1) the 'FCFS' DNE and (2) NADINO's

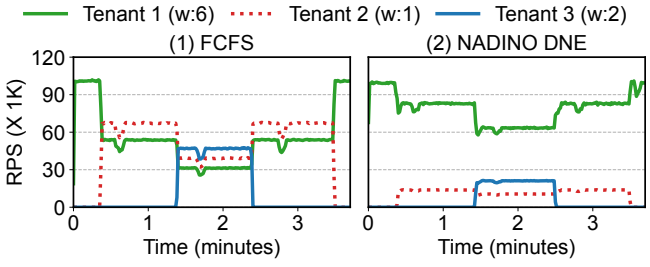

Figure 15. Effect of RDMA network isolation in NADINO: (1) "FCFS" DNE *without* multi-tenancy support; (2) NADINO's DNE *with* multi-tenancy support.

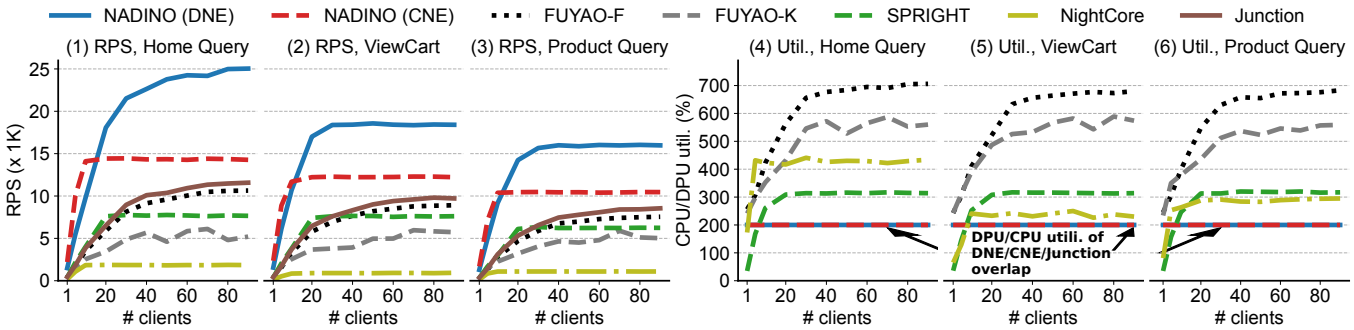

Figure 16. Online boutique results for different chains: Fig. (1) to (3): RPS; Fig. (4) to (6): Efficiency of offloading (CPU and DPU utilization). NADINO (DNE) shows DPU utilization. Others show CPU utilization.

DNE with multi-tenancy support. The FCFS DNE simply processes requests in order, favoring tenants that send requests earlier rather than distributing bandwidth according to predefined weights. This leads to *unfair* bandwidth sharing, as evidenced by the starvation effect observed for Tenant-1 once additional bursty tenants join. In contrast, NADINO's DNE shows improved fairness by precisely scheduling traffic based on their allocated weights when different tenants are competing for bandwidth. All received a weighted fair share in a controlled manner: Upon Tenant-2's arrival, it received 15K RPS, while Tenant-1's RPS decreased from 115K to 90K - precisely maintaining the 1:6 ratio based on their weights. When Tenant-3 joined, the bandwidth distribution adjusts to 65K, 11K, and 22K for Tenant-1, 2, and 3 respectively, again aligning exactly with their weights. Tenant 1 is consistently processed at a higher rate. Additionally, since NADINO supports multi-tenancy via a userspace software solution, it is easy for users to apply workload-specific optimizations by customizing policies in DNE. Additional scalability results with up to six tenants, demonstrating sustained bandwidth saturation and fairness, are provided in Appendix A.

4.3 Putting It All Together: End-to-end Evaluation

Workload Setup: We use the representative Online Boutique [17], a microservices-based application, including 10 functions and offers up to 6 different function chains. Refer to [17] for the function call graph chains of the Online Boutique. We focus on a distributed setup with two worker nodes to holistically evaluate the benefits of seamlessly integrating inter-node RDMA transfers with intra-node shared memory in NADINO: We place the potential hotspot functions (Frontend, Checkout, and Recommendation) on one node while placing the remaining functions on a second node. We use the wrk [18] as the load generator.

Baseline Configurations: We compare NADINO with three advanced serverless data plane designs: SPRIGHT [78], NightCore [42] and FUYAO [64]. All these approaches use local shared memory processing. Since NightCore's network engine is implemented only for intra-node communication, it lacks support for inter-function communication across nodes within a function chain. Therefore, we configure NightCore to run all functions on a single node. In contrast, SPRIGHT supports inter-function communication across nodes (via the kernel protocol stack), which we use as a fairer inter-node baseline. FUYAO uses RDMA for inter-node communication but relies on one-sided writes with receiver-side copying. Note that although FUYAO could be re-implemented with two-sided RDMA, it still maintains separate memory pools for intra/intra-node communication, incurring unnecessary data copies between individual memory pools. Similar to NADINO, each of these approaches incorporates a node-wide network engine-like component to facilitate data movement in and out of the local memory pool. We further compare NADINO against Junction [36] to quantify the cost of software-based kernel-bypass networking. Junction uses the same cross-node function placement as NADINO. Junction function relies on its built-in kernel-bypass TCP/IP stack for inter-function data transfers, both within and across nodes.

Due to the severely poor overload behavior of kernel-based cluster ingress (*K-Ingress*) identified in §4.1.3, we use the F-stack NGINX (*F-Ingress*) as the HTTP/TCP cluster ingress for SPRIGHT and Junction. We additionally evaluate FUYAO with both *K-Ingress* (denoted as "`FUYAO-K`") and *F-Ingress* (denoted as "`FUYAO-F`") in §4.1.3. NightCore relies on its built-in kernel-based cluster ingress. To quantify the benefit of DPU offloading, we additionally run NADINO's Network Engine on the CPU (version called "NADINO (CNE)") for an apples-to-apples comparison.

Performance Analysis: Fig. 16 (1) shows RPS of three chains ('Home Query', 'ViewCart', 'Product Query'), each of which incur more than 11 data exchanges between functions. NightCore's RPS remains 5.1× ~20.9× lower than NADINO even though it places all functions on the same node to fully exploit shared memory processing. We attribute this gap to the duplicated overhead of HTTP/TCP processing at both the worker node and cluster ingress. SPRIGHT, working with DPDK-based *F-Ingress*, achieves up to 8.6× RPS improvement vs. NightCore. The same observation can be made by comparing `FUYAO-F` and `FUYAO-K`, which aligns with our findings in §4.1.3. Both, NADINO (CNE) and NADINO (DNE) take one step further: terminating TCP connections at the ingress and converting traffic to RDMA for fast transmission, which

Table 2. Average latency (in ms) of online boutique chains.

	Home Query			View Cart			Product Query		
# clients	**20**	**60**	**80**	**20**	**60**	**80**	**20**	**60**	**80**
NADINO (DNE)	***1.12***	***2.55***	***3.19***	2.07	***3.35***	***4.36***	2.07	***3.92***	***5.18***
NADINO (CNE)	1.43	4.39	5.62	***1.85***	5.14	6.50	***2.06***	6.04	7.63
FUYAO-F	3.53	5.96	7.53	3.43	7.16	9.02	4.26	8.23	10.7
FUYAO-K	4.88	8.28	20.5	4.95	9.22	10.2	5.33	9.46	11.4
Junction	3.06	5.20	6.68	3.07	6.28	7.86	3.74	7.31	9.47
SPRIGHT	2.66	7.78	10.4	2.69	7.95	10.6	3.27	9.62	12.8
NightCore	10.8	32.4	42.8	22.2	65.1	89.5	18.4	55.2	73.4

significantly reduce transport protocol processing overhead. Jointly working with the lock-free inter-node data transfers, using two-sided RDMA, NADINO (DNE) achieves 2.1× ~4.1× improved RPS compared to FUYAO-F and 2.4× ~4.1× improved RPS compared to SPRIGHT. Junction's RPS is over 47% lower than NADINO (DNE) and over 17% lower than NADINO (CNE) due to software-based protocol processing overhead, which is duplicated for inter-function communication. Just like its lower RPS, Junction also shows higher latency than NADINO (Table 2).

NADINO's DNE also outperforms NADINO (CNE) (1.3× ~1.8× higher RPS) when handling more than 20 clients: since CNE co-locates user functions on the CPU, DOCA Comch is no longer needed. We change the CNE to use eBPF's SK_MSG to directly interact with the user functions for intra-node IPC. However, SK_MSG's interrupt-driven model begins to experience interrupt processing load [72] of CNE at high concurrency (also the single CNE is shared by SK_MSG of many functions), throttling CNE's I/O performance. In contrast, DNE design offloads the I/O handling of user functions to the DPU, eliminating excessive interrupts and sustaining higher throughput. Under light load (fewer than 20 clients), CNE attains slightly lower latency than DNE, as interrupt overhead is minimal, and the CPU's general-purpose cores yield better responsiveness (see Table 2). Both NADINO DNE and CNE latency have significantly lower than the other alternatives.

4.3.1 Efficiency of DPU Offloading. To assess the efficiency of DPU-offloaded network engine in NADINO, we analyze the CPU/DPU core utilization at the corresponding load level. For comparison, all other alternatives run their network engines on the CPU, similar to NADINO's CNE.

Fig. 16 (4) - (6) shows the efficiency results. FUYAO (both FUYAO-K and FUYAO-F) constantly saturates its assigned CPU core (more than 500%) at high load. This is due to its reliance on one-sided write, which requires the receiver to continuously poll for data arrivals and takes up one core each on every worker node. In addition, FUYAO's reliance on kernel networking to terminate HTTP/TCP traffic from external clients and receiver-side copying causes FUYAO's variants to have up to 3.5× more CPU utilization than NADINO (CNE). NADINO (CNE) also has lower CPU utilization, by up to 1.58× compared to SPRIGHT and up to 2.17× compared to NightCore, owing to its use of RDMA. Junction relies on dedicating one full CPU core per node solely for scheduling, which reaches 100% utilization but does not contribute directly to packet processing.

When we compare NADINO (DNE) and NADINO (CNE), both operate in a run-to-completion busy loop, maintaining 100% utilization of the assigned DPU/CPU core(s) regardless of the load (a total of 200% since we use two worker nodes - thus two DNEs/CNEs). However, NADINO (DNE) consistently delivers superior performance than NADINO (CNE) and all other CPU-based alternatives. This demonstrates that, through careful optimization (two-sided RDMA, run-to-completion loop and cross-processor shared memory), we can unleash the power of DPU even though its core is much less capable than the CPU core.

The performance and DPU utilization results also suggest NADINO is more energy efficient than the CPU-only baseline: NADINO's DNE runs on ARM-based DPU cores (Armv8 A72), which consume less power than x86 CPU cores due to a more streamlined instruction set and lower frequency (up to 2.5Ghz on DPU cores vs. up to 3.7Ghz on CPU cores in our testbed). This highlights the effectiveness of leveraging a dedicated DPU for network processing, freeing up valuable CPU resources while maximizing overall system efficiency (we leave direct energy measurements to future work).

5 Conclusion

NADINO demonstrated the effectiveness of RDMA-capable DPU offloading in multi-tenant serverless clouds for improving data plane performance while reducing CPU usage. By combining two-sided RDMA with local shared memory processing and early termination of HTTP/TCP protocol at the cluster ingress, NADINO achieves up to 20.9× RPS improvement and 21× latency reduction compared to state-of-the-arts serverless data planes, using either RDMA or shared memory, when serving a complex online boutique workload.

With NADINO, we show that these SmartNIC-like DPUs are NOT in name only but have the ability to reshape the serverless data plane by delivering zero-copy's efficiency and performance, while enforcing network isolation at limited extra cost. Through careful optimization of the CPU-DPU datapath (via CPU-DPU memory sharing) and the streamlined run-to-completion packet processing pipeline, a DPU-enabled network engine can largely outperform a CPU-based network engine (up to 1.8× RPS improvement in an apples-to-apples comparison) when executing key data plane tasks in NADINO, including inter-node RDMA data transfers and managing multi-tenancy, while freeing up 7 CPU cores and actively using only two wimpy DPU cores.

Acknowledgments

We thank the U.S. NSF for their generous support with JUNO-3 grant 2210379. We thank our shepherd, Dr. Anjo Vahldiek-Oberwagner, and the anonymous reviewers for their valuable suggestions and comments.

References

[1] 2023. The Istio service mesh. https://istio.io/v1.16/about/service-mesh/. [online].

[2] 2024. BPF_PROG_TYPE_SK_MSG. https://ebpf-docs.dylanreimerink.nl/linux/program-type/BPF_PROG_TYPE_SK_MSG/. [online].

[3] 2024. Cilium Service Mesh. https://cilium.io/use-cases/service-mesh/. [online].

[4] 2024. Knative. https://knative.dev/docs/serving/architecture/. [online].

[5] 2024. Netronome Agilio SmartNICs. https://netronome.com/. [online].

[6] 2024. NVIDIA BlueField Networking Platform. https://www.nvidia.com/en-us/networking/products/data-processing-unit/. [online].

[7] 2024. What is FaaS? https://www.ibm.com/topics/faas. [online].

[8] 2025. A Detailed Explanation about Alibaba Cloud CIPU. https://www.alibabacloud.com/blog/a-detailed-explanation-about-alibaba-cloud-cipu_599183. [online].

[9] 2025. Broadcom Stingray SmartNIC Accelerates Baidu Cloud Services. https://investors.broadcom.com/news-releases/news-release-details/broadcom-stingray-smartnic-accelerates-baidu-cloud-services. [online].

[10] 2025. DOCA Comch. https://docs.nvidia.com/doca/sdk/doca+comch/index.html. [online].

[11] 2025. DOCA Progress Engine. https://docs.nvidia.com/doca/sdk/doca+core/index.html#src-3725352556_id-.DOCACorev3.0.0-DOCAProgressEngine. [online].

[12] 2025. DPDK Environment Abstraction Layer (EAL) Library. https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html. [online].

[13] 2025. DPDK Multi-process Support. https://doc.dpdk.org/guides/prog_guide/multi_proc_support.html. [online].

[14] 2025. F-Stack. https://github.com/F-Stack/f-stack.git. [online].

[15] 2025. NVIDIA DOCA Software Framework. https://developer.nvidia.com/networking/doca. [online].

[16] 2025. Offloading and isolating data center workloads with Bluefield DPU. https://developer.nvidia.com/blog/offloading-and-isolating-data-center-workloads-with-bluefield-dpu/. [online].

[17] 2025. Online Boutique. https://github.com/GoogleCloudPlatform/microservices-demo. [online].

[18] 2025. wrk - a HTTP benchmarking tool. https://github.com/wg/wrk. [online].

[19] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[20] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. 2023. Empowering Azure Storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 49–67. https://www.usenix.org/conference/nsdi23/presentation/bai

[21] Amanda Baran, Jacob Nelson-Slivon, Lewis Tseng, and Roberto Palmieri. 2024. ALock: Asymmetric Lock Primitive for RDMA Systems. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures* (Nantes, France) *(SPAA '24)*. Association for Computing Machinery, New York, NY, USA, 15–26. https://doi.org/10.1145/3626183.3659977

[22] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 49–65. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/belay

[23] Sol Boucher, Anuj Kalia, David G. Andersen, and Michael Kaminsky. 2018. Putting the "Micro" Back in Microservice. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 645–650. https://www.usenix.org/conference/atc18/presentation/boucher

[24] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan R. K. Ports. 2021. PRISM: Rethinking the RDMA Interface for Distributed Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 228–242. https://doi.org/10.1145/3477132.3483587

[25] Idan Burstein. 2021. Nvidia Data Center Processing Unit (DPU) Architecture. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. 1–20. https://doi.org/10.1109/HCS52781.2021.9567066

[26] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 65–77. https://doi.org/10.1145/3452296.3472888

[27] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. 2020. Wukong: a scalable and locality-enhanced framework for serverless parallel computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3419111.3421286

[28] Jingrong Chen, Yongji Wu, Shihan Lin, Yechen Xu, Xinhao Kong, Thomas Anderson, Matthew Lentz, Xiaowei Yang, and Danyang Zhuo. 2023. Remote Procedure Call as a Managed System Service. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 141–159. https://www.usenix.org/conference/nsdi23/presentation/chen-jingrong

[29] Qiong Chen, Jianmin Qian, Yulin Che, Ziqi Lin, Jianfeng Wang, Jie Zhou, Licheng Song, Yi Liang, Jie Wu, Wei Zheng, Wei Liu, Linfeng Li, Fangming Liu, and Kun Tan. 2024. YuanRong: A Production General-purpose Serverless System for Distributed Applications in the Cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 843–859. https://doi.org/10.1145/3651890.3672216

[30] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. 2020. λ-NIC: Interactive Serverless Compute on Programmable SmartNICs. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. 67–77. https://doi.org/10.1109/ICDCS47774.2020.00029

[31] Tianyi Cui, Chenxingyu Zhao, Wei Zhang, Kaiyuan Zhang, and Arvind Krishnamurthy. 2024. Laconic: Streamlined Load Balancers

for SmartNICs. arXiv:2403.11411 [cs.NI]

[32] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 401–414. https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi{ć}

[33] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 467–481. https://doi.org/10.1145/3373376.3378512

[34] F5 Networks, Inc. 2022. NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy. https://www.nginx.com/. [online].

[35] 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, Renton, WA, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[36] Joshua Fried, Gohar Irfan Chaudhry, Enrique Saurez, Esha Choukse, Inigo Goiri, Sameh Elnikety, Rodrigo Fonseca, and Adam Belay. 2024. Making Kernel Bypass Practical for the Cloud with Junction. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 55–73. https://www.usenix.org/conference/nsdi24/presentation/fried

[37] Alexander Fuerst and Prateek Sharma. 2021. FaasCache: keeping serverless computing alive with greedy-dual caching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 386–400. https://doi.org/10.1145/3445814.3446757

[38] Jian Gao, Youyou Lu, Minhui Xie, Qing Wang, and Jiwu Shu. 2023. Citron: Distributed Range Lock Management with One-sided RDMA. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. USENIX Association, Santa Clara, CA, 297–314. https://www.usenix.org/conference/fast23/presentation/gao

[39] Raffaele Giannessi, Alessandro Biondi, and Alessandro Biasci. 2024. RT-Mimalloc: A New Look at Dynamic Memory Allocation for Real-Time Systems. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 173–185. https://doi.org/10.1109/RTAS61025.2024.00022

[40] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan M. G. Wassel, Zhehua Wu, Sunghwan Yoo, Raghuraman Balasubramanian, Prashant Chandra, Michael Cutforth, Peter Cuy, David Decotigny, Rakesh Gautam, Alex Iriza, Milo M. K. Martin, Rick Roy, Zuowei Shen, Ming Tan, Ye Tang, Monica Wong-Chan, Joe Zbiciak, and Amin Vahdat. 2022. Aquila: A unified, low-latency fabric for datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1249–1266. https://www.usenix.org/conference/nsdi22/presentation/gibson

[41] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. 2020. MasQ: RDMA for Virtual Private 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* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3387514.3405849

[42] Zhipeng Jia and Emmett Witchel. 2021. Nightcore: Efficient and Scalable Serverless Computing for Latency-Sensitive, Interactive Microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 152–166. https://doi.org/10.1145/3445814.3446701

[43] Chao Jin, Zili Zhang, Xingyu Xiang, Songyun Zou, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Ditto: Efficient Serverless Analytics with Elastic Parallelism. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 406–419. https://doi.org/10.1145/3603269.3604816

[44] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. arXiv:1902.03383 [cs.OS]

[45] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[46] Svilen Kanev, Sam Likun Xi, Gu-Yeon Wei, and David Brooks. 2017. Mallacc: Accelerating Memory Allocation. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 33–45. https://doi.org/10.1145/3037697.3037736

[47] Mikhail Khalilov, Marcin Chrapek, Siyuan Shen, Alessandro Vezzu, Thomas Benz, Salvatore Di Girolamo, Timo Schneider, Daniele De Sensi, Luca Benini, and Torsten Hoefler. 2024. OSMOSIS: Enabling Multi-Tenancy in Datacenter SmartNICs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 247–263. https://www.usenix.org/conference/atc24/presentation/khalilov

[48] Daehyeok Kim, Tianlong Yu, Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Shachar Raindel, Chuanxiong Guo, Vyas Sekar, and Srinivasan Seshan. 2019. FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 113–126. https://www.usenix.org/conference/nsdi19/presentation/kim

[49] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. 2021. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 756–771. https://doi.org/10.1145/3477132.3483565

[50] Taehyun Kim, Deondre Martin Ng, Junzhi Gong, Youngjin Kwon, Minlan Yu, and KyoungSoo Park. 2023. Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 275–292. https://www.usenix.org/conference/nsdi23/presentation/kim-taehyun

[51] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security*

and Privacy (SP). 1–19. https://doi.org/10.1109/SP.2019.00002

[52] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R. Lebeck, and Danyang Zhuo. 2023. Understanding RDMA Microarchitecture Resources for Performance Isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 31–48. https://www.usenix.org/conference/nsdi23/presentation/kong

[53] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding Performance Anomalies in RDMA Subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 287–305. https://www.usenix.org/conference/nsdi22/presentation/kong

[54] Adithya Kumar, Anand Sivasubramaniam, and Timothy Zhu. 2023. SplitRPC: A Control + Data Path Splitting RPC Stack for ML Inference Serving. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 2, Article 30 (May 2023), 26 pages. https://doi.org/10.1145/3589974

[55] Yanfang Le, Brent Stephens, Arjun Singhvi, Aditya Akella, and Michael M. Swift. 2018. RoGUE: RDMA over Generic Unconverged Ethernet. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 225–236. https://doi.org/10.1145/3267809.3267826

[56] Collin Lee and John Ousterhout. 2019. Granular Computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 149–154. https://doi.org/10.1145/3317550.3321447

[57] Junru Li, Youyou Lu, Qing Wang, Jiazhen Lin, Zhe Yang, and Jiwu Shu. 2022. AlNiCo: SmartNIC-accelerated Contention-aware Request Scheduling for Transaction Processing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 951–966. https://www.usenix.org/conference/atc22/presentation/li-junru

[58] Ming Li, Wenyan Lu, Hanyue Lin, Jingya Wu, Yu Zhang, and Guihai Yan. 2023. FlatProxy: A DPU-centric Service Mesh Architecture for Hyperscale Cloud-native Application. arXiv:2312.01297 [cs.DC]

[59] Qiang Li, Yixiao Gao, Xiaoliang Wang, Haonan Qiu, Yanfang Le, Derui Liu, Qiao Xiang, Fei Feng, Peng Zhang, Bo Li, Jianbo Dong, Lingbo Tang, Hongqiang Harry Liu, Shaozong Liu, Weijie Li, Rui Miao, Yaohui Wu, Zhiwu Wu, Chao Han, Lei Yan, Zheng Cao, Zhongjie Wu, Chen Tian, Guihai Chen, Dennis Cai, Jinbo Wu, Jiaji Zhu, Jiesheng Wu, and Jiwu Shu. 2023. Flor: An Open High Performance RDMA Framework Over Heterogeneous RNICs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 931–948. https://www.usenix.org/conference/osdi23/presentation/li-qiang

[60] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. 2022. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 53–68. https://www.usenix.org/conference/atc22/presentation/li-zijun-rund

[61] Jiaxin Lin, Adney Cardoza, Tarannum Khan, Yeonju Ro, Brent E. Stephens, Hassan Wassel, and Aditya Akella. 2023. RingLeader: Efficiently Offloading Intra-Server Orchestration to NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1293–1308. https://www.usenix.org/conference/nsdi23/presentation/lin

[62] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259. https://www.usenix.org/conference/osdi20/presentation/lin

[63] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 973–990. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[64] Guowei Liu, Laiping Zhao, Yiming Li, Zhaolin Duan, Sheng Chen, Yitao Hu, Zhiyuan Su, and Wenyu Qu. 2024. FUYAO: DPU-enabled Direct Data Transfer for Serverless Computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 431–447. https://doi.org/10.1145/3620666.3651327

[65] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading distributed applications onto smartNICs using iPipe. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 318–333. https://doi.org/10.1145/3341302.3342079

[66] Jiaqi Lou, Xinhao Kong, Jinghan Huang, Wei Bai, Nam Sung Kim, and Danyang Zhuo. 2024. Harmonic: Hardware-assisted RDMA Performance Isolation for Public Clouds. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1479–1496. https://www.usenix.org/conference/nsdi24/presentation/lou

[67] Fangming Lu, Xingda Wei, Zhuobin Huang, Rong Chen, Minyu Wu, and Haibo Chen. 2024. Serialization/Deserialization-free State Transfer in Serverless Workflows. In *Proceedings of the Nineteenth European Conference on Computer Systems* (<conf-loc>, <city>Athens</city>, <country>Greece</country>, </conf-loc>) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 132–147. https://doi.org/10.1145/3627703.3629568

[68] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 412–426. https://doi.org/10.1145/3472883.3487003

[69] 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

[70] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. 2021. Gimbal: enabling multi-tenant storage disaggregation on SmartNIC JBOFs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 106–122. https://doi.org/10.1145/3452296.3472940

[71] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting network support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 313–326. https://doi.org/10.1145/3230543.3230557

[72] Jeffrey C. Mogul and K. K. Ramakrishnan. 1997. Eliminating receive livelock in an interrupt-driven kernel. *ACM Trans. Comput. Syst.* 15, 3 (Aug. 1997), 217–252. https://doi.org/10.1145/263326.263335

[73] Sumit Kumar Monga, Sanidhya Kashyap, and Changwoo Min. 2021. Birds of a Feather Flock Together: Scaling RDMA RPCs with Flock. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 212–227. https://doi.org/10.1145/3477132.3483576

[74] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 77–92. https://www.usenix.org/conference/nsdi20/presentation/moon

[75] Joel Nider and Alexandra (Sasha) Fedorova. 2021. The last CPU. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Ann Arbor, Michigan) *(HotOS '21)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3458336.3465291

[76] Federico Parola, Shixiong Qi, Anvaya B. Narappa, K. K. Ramakrishnan, and Fulvio Risso. 2024. SURE: Secure Unikernels Make Serverless Computing Rapid and Efficient. In *Proceedings of the 2024 ACM Symposium on Cloud Computing* (Redmond, WA, USA) *(SoCC '24)*. Association for Computing Machinery, New York, NY, USA, 668–688. https://doi.org/10.1145/3698038.3698558

[77] Dinglan Peng, Congyu Liu, Tapti Palit, Anjo Vahldiek-Oberwagner, Mona Vij, and Pedro Fonseca. 2025. Pegasus: Transparent and Unified Kernel-Bypass Networking for Fast Local and Remote Communication. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 360–378. https://doi.org/10.1145/3689031.3696083

[78] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and K. K. Ramakrishnan. 2022. SPRIGHT: Extracting the Server from Serverless Computing! High-Performance EBPF-Based Event-Driven, Shared-Memory Processing. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 780–794. https://doi.org/10.1145/3544216.3544259

[79] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. 2021. ReDMArk: Bypassing RDMA Security Mechanisms. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 4277–4292. https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger

[80] Alireza Sahraei, Soteris Demetriou, Amirali Sobhgol, Haoran Zhang, Abhigna Nagaraja, Neeraj Pathak, Girish Joshi, Carla Souza, Bo Huang, Wyatt Cook, Andrii Golovei, Pradeep Venkat, Andrew Mcfague, Dimitrios Skarlatos, Vipul Patel, Ravinder Thind, Ernesto Gonzalez, Yun Jin, and Chunqiang Tang. 2023. XFaaS: Hyperscale and Low Cost Serverless Functions at Meta. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 231–246. https://doi.org/10.1145/3600006.3613155

[81] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. 2021. Xenic: SmartNIC-Accelerated Distributed Transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 740–755. https://doi.org/10.1145/3477132.3483555

[82] Vishwanath Seshagiri, Abhinav Gupta, Vahab Jabrayilov, Avani Wildani, and Kostis Kaffes. 2024. Rethinking the Networking Stack for Serverless Environments: A Sidecar Approach. In *Proceedings of the 2024 ACM Symposium on Cloud Computing* (Redmond, WA, USA) *(SoCC '24)*. Association for Computing Machinery, New York, NY, USA, 213–222. https://doi.org/10.1145/3698038.3698561

[83] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 87–102. https://www.usenix.org/conference/nsdi22/presentation/shashidhara

[84] Simon Shillaker and Peter Pietzuch. 2020. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 419–433. https://www.usenix.org/conference/atc20/presentation/shillaker

[85] M. Shreedhar and George Varghese. 1995. Efficient fair queueing using deficit round robin. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Cambridge, Massachusetts, USA) *(SIGCOMM '95)*. Association for Computing Machinery, New York, NY, USA, 231–242. https://doi.org/10.1145/217382.217453

[86] Junyi Shu, Kun Qian, Ennan Zhai, Xuanzhe Liu, and Xin Jin. 2024. Burstable Cloud Block Storage with Data Processing Units. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 783–799. https://www.usenix.org/conference/osdi24/presentation/shu

[87] Enge Song, Yang Song, Chengyun Lu, Tian Pan, Shaokai Zhang, Jianyuan Lu, Jiangu Zhao, Xining Wang, Xiaomin Wu, Minglan Gao, Zongquan Li, Ziyang Fang, Biao Lyu, Pengyu Zhang, Rong Wen, Li Yi, Zhigang Zong, and Shunmin Zhu. 2024. Canal Mesh: A Cloud-Scale Sidecar-Free Multi-Tenant Service Mesh Architecture. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 860–875. https://doi.org/10.1145/3651890.3672221

[88] Sayantan Sur, Hyun-Wook Jin, Lei Chai, and Dhabaleswar K. Panda. 2006. RDMA read based rendezvous protocol for MPI over InfiniBand: design alternatives and benefits. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, New York, USA) *(PPoPP '06)*. Association for Computing Machinery, New York, NY, USA, 32–39. https://doi.org/10.1145/1122971.1122978

[89] Ariel Szekely, Adam Belay, Robert Morris, and M. Frans Kaashoek. 2024. Unifying serverless and microservice workloads with SigmaOS. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) *(SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 385–402. https://doi.org/10.1145/3694715.3695947

[90] Shelby Thomas, Lixiang Ao, Geoffrey M. Voelker, and George Porter. 2020. Particle: ephemeral endpoints for serverless networking. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 16–29. https://doi.org/10.1145/3419111.3421275

[91] Shin-Yeh Tsai and Yiying Zhang. 2017. LITE Kernel RDMA Support for Datacenter Applications. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 306–324. https://doi.org/10.1145/3132747.3132762

[92] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. 2020. InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. USENIX Association, Santa Clara, CA, 267–281. https://www.usenix.org/conference/fast20/presentation/wang-ao

[93] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong

Guo. 2023. SRNIC: A Scalable Architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1–14. https://www.usenix.org/conference/nsdi23/presentation/wang-zilong

[94] Zilong Wang, Xinchen Wan, Chaoliang Zeng, and Kai Chen. 2023. Accurate and Scalable Rate Limiter for RDMA NICs. In *Proceedings of the 7th Asia-Pacific Workshop on Networking* (Hong Kong, China) *(APNet '23)*. Association for Computing Machinery, New York, NY, USA, 15–20. https://doi.org/10.1145/3600061.3600078

[95] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. 2023. Characterizing Off-path SmartNIC for Accelerating Distributed Systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 987–1004. https://www.usenix.org/conference/osdi23/presentation/wei-smartnic

[96] Xingda Wei, Fangming Lu, Rong Chen, and Haibo Chen. 2022. KRCORE: A Microsecond-scale RDMA Control Plane for Elastic Computing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 121–136. https://www.usenix.org/conference/atc22/presentation/wei

[97] Yunming Xiao, Diman Zad Tootaghaj, Aditya Dhakal, Lianjie Cao, Puneet Sharma, and Aleksandar Kuzmanovic. 2024. Conspirator: SmartNIC-Aided Control Plane for Distributed ML Workloads. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 767–784. https://www.usenix.org/conference/atc24/presentation/xiao

[98] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. 2022. Bedrock: Programmable Network Support for Secure RDMA Systems. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2585–2600. https://www.usenix.org/conference/usenixsecurity22/presentation/xing

[99] Dong Young Yoon, Mosharaf Chowdhury, and Barzan Mozafari. 2018. Distributed Lock Management with RDMA: Decentralization without Starvation. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1571–1586. https://doi.org/10.1145/3183713.3196890

[100] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. 2023. Following the Data, Not the Function: Rethinking Function Orchestration in Serverless Computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1489–1504. https://www.usenix.org/conference/nsdi23/presentation/yu

[101] Qingyang Zeng, Kaiyu Hou, Xue Leng, and Yan Chen. 2024. DirectFaaS: A Clean-Slate Network Architecture for Efficient Serverless Chain Communications. In *Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 2759–2767. https://doi.org/10.1145/3589334.3645333

[102] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. 2021. The Demikernel Datapath OS Architecture for Microsecond-scale Datacenter Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 195–211. https://doi.org/10.1145/3477132.3483569

[103] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. 2022. Justitia: Software Multi-Tenancy in Hardware Kernel-Bypass Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1307–1326. https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen

[104] Chenxingyu Zhao, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2025. White-Boxing RDMA with Packet-Granular Software Control. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 427–449. https://www.usenix.org/conference/nsdi25/presentation/zhao-chenxingyu

[105] Weiyue Zhao, Jingya Wu, Wenyan Lu, Xiaowei Li, and Guihai Yan. 2024. TianMen: a DPU-based storage network offloading structure for disaggregated datacenters. In *Proceedings of the 2024 ACM Symposium on Cloud Computing* (Redmond, WA, USA) *(SoCC '24)*. Association for Computing Machinery, New York, NY, USA, 689–703. https://doi.org/10.1145/3698038.3698528

[106] Yang Zhou, Mark Wilkening, James Mickens, and Minlan Yu. 2024. SmartNIC Security Isolation in the Cloud with S-NIC. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 851–869. https://doi.org/10.1145/3627703.3650071

[107] Zhuangzhuang Zhou, Vaibhav Gogte, Nilay Vaish, Chris Kennelly, Patrick Xia, Svilen Kanev, Tipp Moseley, Christina Delimitrou, and Parthasarathy Ranganathan. 2024. Characterizing a Memory Allocator at Warehouse Scale. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 192–206. https://doi.org/10.1145/3620666.3651350

A Scalability of Multi-tenancy Support

To further evaluate the scalability of NADINO's multi-tenancy support, we extend the setup to 6 tenants, all assigned equal weights. Each tenant issues client/server requests across two worker nodes, generating inter-node RDMA traffic through the NADINO DNE. The experiment begins with a single tenant, and a new tenant is added roughly every 30 seconds, until all 6 tenants are active. Similarly, after every 30 seconds, a tenant is removed. All other experiment settings are identical to §4.2.

All the tenants exhibit similar bandwidth shares as the competing tenants at a given time (as shown in Fig. 17 (1)), as with the 3-tenant case, indicating that fairness is preserved even with higher tenant counts. Fig. 17 (2) shows that the aggregate RPS stays near 110K (the maximum throughput of the single DPU core we have allocated), showing that the DNE continues to saturate the available bandwidth as tenants scale from 3 to 6.

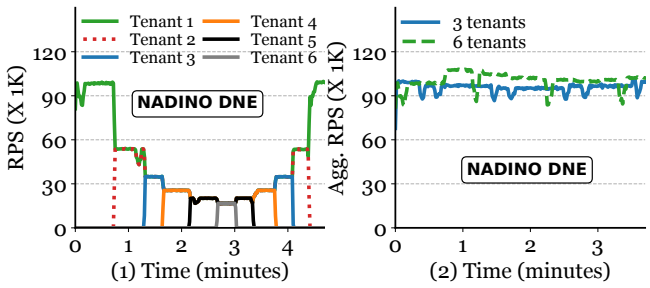

Figure 17. Scalability of NADINO's multi-tenancy support. (1) NADINO's DNE fairly shares the bandwidth across competing tenants. (2) The aggregated RPS of 3 versus 6 tenants.