FlexDriver: A Network Driver for Your Accelerator

Haggai Eran
NVIDIA and Technion
Yokne'am Ilit, Israel

Maxim Fudim
NVIDIA
Yokne'am Ilit, Israel

Gabi Malka
Technion
Haifa, Israel

Gal Shalom
NVIDIA and Technion
Yokne'am Ilit, Israel

Noam Cohen
NVIDIA
Yokne'am Ilit, Israel

Amit Hermony
NVIDIA
Yokne'am Ilit, Israel

Dotan Levi
NVIDIA
Yokne'am Ilit, Israel

Liran Liss
NVIDIA
Yokne'am Ilit, Israel

Mark Silberstein
Technion
Haifa, Israel

ABSTRACT

We propose a new system design for connecting hardware and FPGA accelerators to the network, allowing the accelerator to directly control commodity Network Interface Cards (NICs) without using the CPU. This enables us to solve the key challenge of leveraging existing NIC hardware offloads such as virtualization, tunneling, and RDMA for accelerator networking. Our approach supports a diverse set of use cases, from direct network access for disaggregated accelerators to inline-acceleration of the network stack, all without the complex networking logic in the accelerator.

To demonstrate the feasibility of this approach, we build **FlexDriver (FLD)**, an on-accelerator hardware module that implements a NIC data-plane driver. Our main technical contribution is a mechanism that compresses the NIC control structures by two orders of magnitude, allowing FLD to achieve high networking scalability with low die area cost and no bandwidth interference with the accelerator logic.

The prototype for NVIDIA Innova-2 FPGA SmartNICs showcases our design's utility for three different accelerators: a disaggregated LTE cipher, an IP-defragmentation inline accelerator, and an IoT cryptographic-token authentication offload. These accelerators reach 25 Gbps line rate and leverage the NIC for RDMA processing, VXLAN tunneling, and traffic shaping without CPU involvement.

CCS CONCEPTS

• **Hardware → Hardware accelerators**; **Networking hardware**; • **Computer systems organization → Heterogeneous (hybrid) systems**; *Distributed architectures*.

KEYWORDS

accelerator networking, accelerator disaggregation, network function acceleration

ACM Reference Format:
Haggai Eran, Maxim Fudim, Gabi Malka, Gal Shalom, Noam Cohen, Amit Hermony, Dotan Levi, Liran Liss, and Mark Silberstein. 2022. FlexDriver: A Network Driver for Your Accelerator. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3503222.3507776

1 INTRODUCTION

Modern data centers rely on special-purpose accelerators to achieve high performance for AI tasks [17, 75], in networking [4, 32, 99] and storage [4] infrastructures, and also offer general-purpose accelerators to their clients [3, 5, 19, 71].

From the perspective of accelerator developers, these diverse workloads pose two seemingly unrelated requirements. First, there is a growing need to support *direct network connectivity with accelerators*. This is essential, for example, in disaggregated data centers where network-accessible accelerators are pooled together [1, 17, 37], as well as in distributed computing applications such as DNN training [68, 75]. Second, as network growth rate outpaces CPU capacity scaling, data-centers require *inline acceleration of packet processing and network function virtualization applications* to achieve high performance without wasting their tenants' CPU resources [32, 65, 94, 98]. In both scenarios, the *efficiency of the accelerator's interaction with the network* is key to achieving the performance and power goals of the whole system. In this work, we demonstrate a unified architectural approach that achieves this goal.

The primary challenge stems from the complex network stack logic, which involves multiple and diverse software layers traditionally running on the CPU and is assisted by a wealth of hardware offloads in the network adapter (NIC). To either connect to the network or accelerate any of its steps, an accelerator must interoperate with this logic efficiently, under strict performance and architectural constraints.

Today's systems use one of the following three approaches to deal with this challenge:

(a) A *CPU-mediated* design [16, 57] relies on CPU mediation between the accelerator and a network stack running on the host. Sometimes the data path is optimized with direct data placement into accelerator memory, but the host retains the control.

(b) An *accelerator-hosted* design [68, 75] moves the entire physical NIC into the accelerator. It is prevalent in FPGA NICs [33, 103, 125], commercial SmartNICs [46, 73, 122], and in network-attached FPGA accelerators [14, 30, 48, 49, 58, 121]. These feature a full, on-FPGA implementation of the networking stack [97, 102] from the physical layer to the application.

(c) A *Bump-in-the-Wire (BITW)* design, broadly used in FPGA-equipped SmartNICs, such as Microsoft Catapult [32], Innova [69], and Intel's PAC N3000 [45], places an accelerator between the NIC and the network. It shares many of the characteristics of accelerator-hosted design but reuses some NIC components for communicating with the host.

Our analysis of these designs identifies a three-way trade-off between the chip area overheads of the on-accelerator networking layer, host CPU overheads to support accelerator networking, and the NIC-accelerated network stack features made available to accelerators (cf. § 3). The existing approaches can only obtain two out of the three desired qualities (Figure 1).

We present a new system architecture, which explores an improved design point by enabling direct control of a NIC by an accelerator. We demonstrate both how this architecture enables accelerators to use advanced NIC offloads for networking and how existing network stacks can integrate custom inline accelerators, all with a low on-accelerator area footprint and no CPU overheads.

The main idea is to let the accelerator control the NIC via its PCI-Express (PCIe) control interface, thus exposing to the accelerator the same NIC functions as those made available to CPU software by the standard NIC driver. At the core of our design is a hardware module called *FlexDriver (FLD)*, which integrates into the accelerator and operates the NIC's PCIe interfaces independent of the host.

This design supports diverse application scenarios. By using advanced NIC transport offloads, such as RDMA, the accelerator can be easily disaggregated and communicate with other RDMA endpoints in distributed applications. For inline network stack acceleration tasks such as IP defragmentation, the NIC's steering engine can forward packets to the accelerator, integrating the processing steps performed on the accelerator with other NIC offloads. Last, the NIC's flow steering, bandwidth shaping, and packet tagging mechanisms help virtualize accelerators, removing the associated complexity from the accelerator implementation.

While conceptually simple, this idea poses a few challenges which our work addresses.

Driver partitioning. In a nutshell, FLD should implement a NIC driver. However, network drivers are highly sophisticated and device-specific. A verbatim hardware port of the driver logic would not only be difficult to design and maintain but would dramatically increase the FLD area footprint on the accelerator. Instead, we carefully partition the driver such that hardware only implements performance-critical data-plane components, while the control-plane, forming most of the driver complexity, runs on the CPU (§ 4.1).

Memory requirements and scaling. To achieve high performance, the accelerators must store NIC control data structures such as Rx/Tx descriptor rings and network buffers. Unfortunately, our analysis shows that such structures may occupy tens of MiBs of memory, especially for more connections and queues (§ 4.3). If stored in the accelerator's DRAM, however, accessing them from FLD would interfere with the accelerator's memory accesses. Our solution employs a novel hardware mechanism that *generates NIC control structures on-the-fly* in hardware in response to the NIC's PCIe requests, instead of keeping these structures in their original form in memory. This mechanism shrinks the memory requirements by over 100×, allowing the NIC control structures to easily fit into FLD-dedicated on-die memory even when provisioned for 400 Gbps future networks and 2K Rx/Tx queues (§ 5.2).

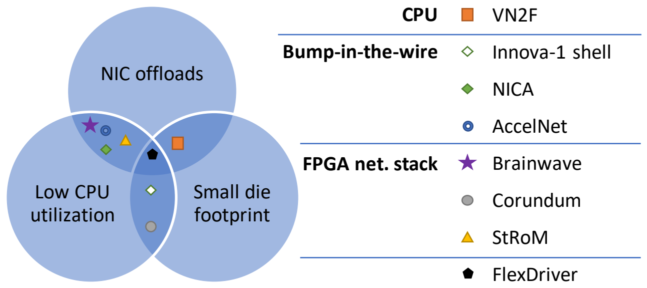


Figure 1: Trade-offs between accelerator area footprint for networking, use of NIC-accelerated features, and CPU overheads.

Software abstractions. FLD retrofits existing RDMA and DPDK software abstractions to simplify integration of FLD-based accelerators with existing systems (§ 5.3).

We prototype FLD and the associated system software by using a 25 Gbps Innova-2 SmartNIC [78], which features an FPGA device and an NVIDIA ConnectX-5 NIC interconnected via PCIe. FLD achieves close to line-rate throughput, utilizing FPGA area on-par or smaller than previously published full network-stack/NIC implementations on FPGA, while supporting much richer network processing functionality. We build and evaluate three sample accelerators to demonstrate the versatility and performance of our approach: a disaggregated accelerator for an LTE cipher exposed via RDMA, an inline IP defragmentation accelerator used in conjunction with the NIC's native VXLAN decapsulation and RSS offloads, and a virtualized IoT authentication accelerator, which leverages the NIC's flow classification and QoS logic.

Finally, our evaluation shows that these accelerators achieve high performance and efficiency while affording relatively simple hardware design, thanks to FLD-enabled NIC offloads.

2 BACKGROUND

We briefly survey the background of NIC offloads and drivers.

2.1 NIC Offloads

NIC vendors implement hardware-accelerated features to offload packet processing tasks from the CPU. These include stateless offloads, such as checksumming, TCP segmentation, receive-side scaling (RSS), and receive flow steering (RFS) [109]. Some NICs implement a transport layer in hardware through remote direct memory access (RDMA [115]).

To let the network stack use the offloads, NICs expose a control interface over PCIe and add metadata to their data-plane PCIe interface.

Multiple offloads can be chained together. For example, receive flow steering can direct traffic to the correct core after IPSec has decrypted a received packet. However, the chaining requires that *all* the tasks in the chain prefix (suffix for transmit) are offloadable ("all-or-nothing offloads" [32]). For example, if the decryption offload is not available on the NIC and needs to be done in software, invoking RFS offload after the packet has left the NIC would be impossible. In FLD, we overcome this problem and allow interleaving packet processing on the accelerator with NIC-offloadable tasks.

2.2 NIC Driver Structure

A NIC driver performs two types of tasks: the control plane tasks, e.g., initialization, teardown, link management, and queue configuration; and the data plane tasks, e.g., packet transmission, posting of receive buffers, and processing of completion notifications.

Driver data-plane tasks commonly use host memory to exchange buffers and completions with the NIC over producer-consumer ring data structures. Typically, the driver transmits packets by storing descriptors in a transmit ring pointing to packet buffers, increasing a producer index, and notifying the NIC using a memory-mapped I/O (MMIO) operation *(doorbell)*. The driver releases the buffers after receiving a completion indication from the NIC. Receiving packets is similar but may skip the MMIO notification.

Transmit and completion descriptors may include metadata accompanying the packet, describing requested offloads (descriptors) or offload results from the NIC (completions). For reliable transports, completions indicate that an entire message (rather than a packet) has been reliably sent or received.

When operating multiple receive queues, reserving data buffers separately for each queue can result in wasted memory due to fragmentation. To mitigate this situation, NICs allow sharing their data buffers through a shared receive queue (SRQ) data structure [40, 108], where multiple receive queues utilize a single memory pool. Similarly, completion queues can be shared among different transmit and receive queues.

2.3 Match-Action Model

Some NICs provide virtualization support, appearing as several virtual NICs (vNICs) that can be dedicated to virtual machines (VMs) for improved performance [24, 47, 59, 79]. The vNICs are connected to the network by an Embedded Switch (eSwitch). For steering packets to the correct VM, NICs use flexible match-action rules [12, 29, 70], managed by the hypervisor. These rules match packets to their destination VM and route them to the correct virtual port (vPort) of the eSwitch. The eSwitch rules can also perform header manipulations such as en/decapsulation or IPSec tunneling.

After a packet arrives at a vPort, the NIC processes it using match-action tables programmed for that port by the guest OS or by a user-space networking application (e.g., using Linux TC filters [39] or DPDK's rte_flow [25]). Packets transmitted by the guest undergo the reverse path, going through guest match-action tables first and then the eSwitch rules.

2.4 PCI-Express

PCI-Express (PCIe) is a high-speed local interconnect commonly used to connect peripheral devices to CPUs [51]. In PCIe, each message includes an address field, and the platform configures each endpoint to be associated with parts of the system's address space through base address registers (BARs) on the endpoint. The term BAR is commonly used to denote the PCIe address space region that belongs to the endpoint itself. PCIe allows endpoints to read and write from other peripherals through peer-to-peer transactions bypassing the CPU, improving performance for a variety of uses [6, 7, 10, 76, 77].

3 ACCELERATOR NETWORKING ARCHITECTURES

Table 1 shows a detailed comparison of several FPGA-based accelerator architectures categorized by their network stack design (Figure 2) using representative prior works.

CPU-Mediated. VN2F [16] exposes an Ethernet interface to the accelerators running on AWS F1 FPGAs while using CPUs to transfer data between a NIC and an FPGA. The accelerator can employ all the features of the CPU network stack, including NIC hardware offloads, while occupying a relatively small area (see Table 1). However, the CPU is involved in every network transaction, limiting scalability, hurting performance, and wasting CPU cycles [22, 60, 91].

Accelerator-Hosted. Corundum [33] and StRoM [103] are full NIC implementations that can be used on FPGA-based accelerators as an integral part of their designs. They are representative of FPGA designs that connect to the network via their integrated Ethernet ports and implement all or parts of the network stack or NIC functionality in programmable logic [46, 73, 122, 125]. Similarly, some machine learning accelerators use integrated NICs [68, 75].

Such solutions do not involve the CPU but use a large die area and programmable logic resources for the boilerplate communication tasks, and usually support fewer network stack features than a NIC. Any feature can be added to the design, but only with an additional area and development effort. For example, adding tunneling support would increase the LUT consumption of Corundum by 30% [96].

Bump-in-the-Wire (BITW). FPGA-based SmartNICs are commonly used for inline acceleration of the network stack [32, 45, 69]. The BITW design combines a NIC ASIC with an FPGA, and it thus saves programmable logic resources by using some of the NIC-supported offloads. NICA [28], for example, leverages the NIC's DMA to the host and SR-IOV.

Unfortunately, as a BITW design connects the accelerator to the NIC's network port, it cannot use some of the NIC offloads efficiently. For example, RDMA-capable NICs implement the transport layer in hardware, but using it requires one to access NIC's PCIe interface. Thus, a BITW accelerator cannot use it to send RDMA messages. Similarly, in inline packet processing applications, ingress packets are first processed by the accelerator, so they cannot utilize NIC offloads such as tunneling decapsulation or IPSec

Table 1: FPGA-based networking architectures.[a] 👍– supported, 🖒– supported only between host and NIC, ✖– unsupported.

			Hardware Utilization				Network features		
Category	Solution	Gbps	LUT	FF	BRAM	URAM	Stateless offloads	Tunneling	Hardware Transport
CPU-mediated	VN2F [16]	10	5.7K	1.1K	233		👍	👍	N/A
Accelerator-hosted	Corundum [33]	25	66.7K	71.7K	239	20	👍	✖	✖
		100	62.4K	76.8K	331	20			
	StRoM [103]	10	92 K	115 K	181		👍	✖	👍
		100	122 K	214 K	402				
BITW	NICA [28]	40	232 K	299 K	584		👍	🖒	👍
	Innova-1 shell [28]	40	169 K	212 K	152		🖒	🖒	🖒
FlexDriver		100[b]	62 K	89 K	79	44	👍	👍	👍

[a]All implementations are on Xilinx UltraScale/UltraScale+ family for a fair comparison.
[b]Our FLD module supports 100 Gbps, but the PCIe link is 50 Gbps, and the port speed is 25 Gbps.

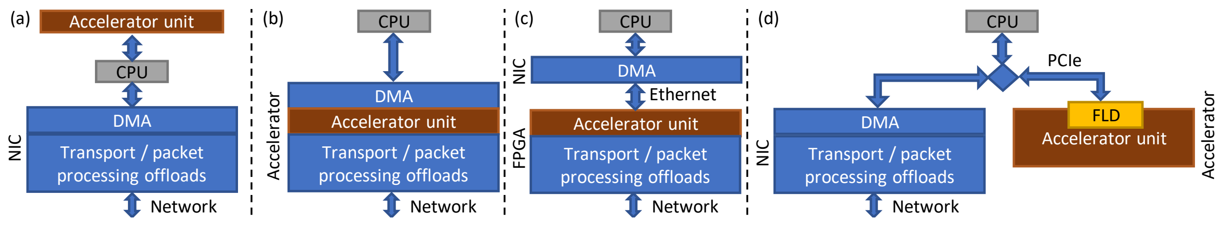


Figure 2: Accelerator-to-network designs: (a) CPU-mediated (b) Accelerator-hosted (c) Bump-in-the-Wire (BITW), and (d) FLD.

decryption. This is why NICA has to reimplement flow steering and QoS traffic classes, among others, leading to its large area footprint.

FlexDriver: This Work. FlexDriver combines the benefits of all three designs: with the direct access to the accelerator from the NIC and from the NIC to the accelerator, it enables inline offloads akin to the BITW and accelerator-hosted designs, while also allowing accelerators to use all NIC offloads, yet without the CPU overheads of a CPU-mediated design.

4 DESIGN CONSIDERATIONS

We aim to achieve the following goals:

(a) enable **low-latency/high-bandwidth** networking support for accelerators;
(b) **minimize CPU overheads** of accelerator networking;
(c) employ unaltered commodity NICs while **utilizing NIC offloads**;
(d) **minimize FLD's area footprint** on the accelerator die;
(e) facilitate integration with simple fixed-function accelerators using an **easy-to-use hardware interface**; and
(f) **reuse existing software**, network stack, and drivers, and interoperate with existing application software, such as RDMA Verbs and DPDK offload interface `rte_flow`.

At a high level, FLD aims to enable accelerators to access NICs over PCIe directly. This requires the functionality that the OS NIC driver traditionally provides. However, porting drivers to hardware verbatim is not viable due to their complexity and the resulting large area footprint of such implementation. Instead, our approach is to partition the driver tasks among the CPU and the accelerator and redesign them to meet stringent hardware resource constraints.

4.1 Division of Labor

To minimize the FLD footprint in the accelerator, we choose to maintain complex control logic on the host CPU while implementing only the performance-critical data-plane functionality in hardware. Thus, FLD handles the following main tasks: (a) managing transmit/receive queues, (b) allocating data buffers, and (c) operating the NIC data-plane PCIe interfaces. We discuss the hardware design in § 5.1. We leave tasks like queue initialization/teardown, connection establishment, and NIC pipeline configuration for software (§ 5.3).

4.2 Location of NIC Control Structures

NIC Control Structures in Host Memory? As explained in § 2.2, a NIC interacts with its CPU driver over a host memory region. But using host memory in the FLD design has several disadvantages. First, NIC–accelerator traffic would compete for PCIe bandwidth with other peripherals, and crucially, with other accelerators using FLD, limiting the solution's scalability. Second, it would pollute CPU caches [31, 74], and compete over memory interfaces and SMP interconnects [105].

NIC Control Structures in Accelerator DRAM? Some accelerators are equipped with local DRAM, which can host the NIC control structures. Compared to host memory, this approach is known to

Table 2: NIC driver memory analysis parameters.

(a) Parameters used in the analysis.

Description	Variable	Value
Bandwidth	B	100 Gbps
Min./max. packet size	$M_{\min}/M_{\max}$	256 B/16 KiB
Lifetime	L_{rx}/L_{tx}	5/25 µs
No. transmit queues	N_q	512
Max. packet rate	$R = \frac{B}{M_{\min}+20\,\text{B}}$	45 Mpps
Min. TX descriptors	$N_{txdesc} = \lceil RL_{tx} \rceil$	1133
Min. RX descriptors	$N_{rxdesc} = \lceil RL_{rx} \rceil$	227
TX Bandwidth × delay	$S_{txbdp} = BL_{tx}$	305 KiB
RX Bandwidth × delay	$S_{rxbdp} = BL_{rx}$	61 KiB

(b) ConnectX/FLD parameters.

Description	Var.	Software	FLD
Tx. descriptor size	S_{txdesc}	64 B	8 B
Rx. descriptor size	S_{rxdesc}	16 B	-
Completion queue entry	S_{cqe}	64 B	15 B
Producer index	S_{pi}	4 B	4 B

improve the accelerator networking performance [22, 91]. However, doing so may cause interference between the accelerator and FLD memory accesses. With DRAM-bandwidth limited [102], adding another memory consumer would not be possible, as is the case for the Innova-2 SmartNIC we use for prototyping FLD. Further, accelerators use a custom memory hierarchy, and accessing it coherently for network management tasks requires additional synchronization operations [81, 91] resulting in suboptimal performance.

Therefore, we store NIC-control data structures on accelerator *dedicated on-chip memory* as part of the FLD module.

4.3 Memory Requirements for Control Structures

Storing NIC control structures on-chip in the structure as in CPU memory might seem to challenge our goal to conserve the accelerator area. Indeed, as we show next, scaling to a large number of queues or high bandwidth would require large amounts of memory in conventional drivers, making the choice of on-die storage for FLD impractical. Later (§5.2), we show how to shrink the memory requirements significantly to make this idea practical.

We use the ConnectX-5 NIC software driver to examine the amount of memory consumed by NIC control structures [27].

Transmit/Receive Buffers. To meet the target bandwidth or packet rate with a single queue, the driver's queue depth has to cover the lifetime of each buffer, multiplied by the expected line rate. For transmission of Ethernet frames, the lifetime includes the time from the buffer allocation until it is transmitted and a completion notification is returned. For RDMA reliable connections, the NIC may retransmit the buffer, and so its lifetime includes the time for the receiver to return an acknowledgment. Receive buffer lifetime spans the time from when the NIC starts filling out a packet buffer until the packet is processed or copied elsewhere by the application or the network stack.

Table 2a shows the parameters of a sample plausible configuration. We assume 100 Gbps line-rate and 25 µs lifetime (matching Azure's reported 99th percentile latency [32]). We use 5 µs latency for receive, assuming the network stack recycles buffers faster. We pick maximal message size large enough to accumulate at least half of the common flow sizes reported in [116]. The number of queues (512) is provisioned for a disaggregated accelerator connected via the NIC, concurrently serving each processor core in a 32 nodes cluster, each node with 16 cores.

Based on these parameters we calculate the minimum number of descriptors $N_{tx/rxdesc}$, and bandwidth-delay product (BDP) $S_{tx/rxbdp}$ representing the lower-bound on the buffer size required to meet line-rate (see Table 2a).

Fragmentation inflates the memory requirements. Basic NIC interfaces for receive-queue management require buffers sized for the maximum packet size. However, some NICs allow more efficient use of memory, and we utilize that in FLD (§ 5.2). For transmit buffers, NICs do not mandate the buffer size, and it is up to the driver software and the network stack to deal with potential fragmentation. Accordingly, many high-performance implementations such as DPDK may use multiple memory pools provisioned for the worst-case traffic pattern with large buffers, thus trading the increased memory consumption for improved performance. Therefore, the resulting buffer sizes are much larger than the BDP. For example, for Tx, S_{txdata} = 17.7 MiB (Table 3) vs. S_{txbdp} = 305 KiB.

Transmit Ring Size. When operating multiple queues, buffers can be shared among them, as not all queues can transmit/receive at the same time, and the overall traffic is bounded by the line rate. Thus, buffer sharing enables scaling the number of queues. Receive buffers can use a shared receive queue and completion queue to share the buffer pool and the descriptor ring itself. Transmit data buffers may also be shared but require a per-queue ring structure. Thus, the memory required for transmit rings grows rapidly with the number of queues, up to S_{txq} = 64 MiB in our example (Table 3).

Other Structures. In addition to the descriptor rings, the driver keeps per-queue producer indices, but with only 4 B the effect on scalability is negligible. The completion queues and the receive ring need only to scale with the overall number of descriptors, as they are shared. We assume one completion queue for all transmit queues and one for receive.

To summarize, our example leads to approximately 85.3 MiB allocated for NIC–driver interaction. For on-die storage on accelerators, such capacity is too large to require. In particular, the Innova-2's FPGA device we use for prototyping has only 10.05 MiB overall available capacity.

In FLD we manage to dramatically reduce memory requirements, by 105× for Table 3's example. We discuss our memory reduction techniques in detail in § 5.2.

4.4 Software Stack

For software, our goal is to extend existing abstractions naturally, but defining suitable abstractions for FLD operation is challenging.

Table 3: Memory analysis for NIC–driver communication with and without FLD optimizations. $f(n) = 2^{\lceil \log_2 n \rceil}$ rounds allocations to a larger power of 2. S_{xlt*} are the translation table sizes, which are <33 KiB. See details in [27].

		Formula to compute		Example		
Description	Var.	Software	FLD	Software	FLD	Shrink ratio
Tx. rings size	S_{txq}	$N_q f(N_{txdesc}) S_{txdesc}$	$f(N_{txdesc}) S_{txdesc} + S_{xltTx}$	64 MiB	32 KiB	×2080
Tx. buffer size	S_{txdata}	$M_{\max} N_{txdesc}$	$2S_{txbdp} + S_{xltData}$	17.7 MiB	643 KiB	×28.2
Rx. buffer size	S_{rxdata}	$M_{\max} N_{rxdesc}$	$2S_{rxbdp}$	3.5 MiB	122 KiB	×29.8
Completion queue size	S_{cq}	$f(N_{txdesc} + N_{rxdesc}) S_{cqe}$		144 KiB	33.75 KiB	×4.27
Rx. ring size	S_{srq}	$f(N_{desc}) S_{rxdesc}$	-	4 KiB	-	
Total producer index size	S_{pitot}	$(N_q + 1) S_{pi}$		2052 B	2052 B	×1
Total		$S_{*data} + S_{txq} + S_{srq} + S_{pitot} + S_{cq}$		85.3 MiB	832.7 KiB	×105

FLD provides a single hardware module design for both inline network acceleration and accelerator networking, yet software support for these two scenarios differs substantially: the former requires marking flows with match-action tables, whereas the latter needs setting up an RDMA Queue-Pair (QP) on the NIC.

For inline acceleration, existing packet processing applications use the match-action tables abstraction to define how the NIC handles ingress and egress packets. We seek to extend this abstraction with new acceleration actions, which move packets to the accelerator and back to the match-action pipeline behind the scenes. RDMA applications use the QP notion to define both a transport endpoint and a NIC software interface, while with FLD, we need them separated.

We discuss the software design in § 5.3.

4.5 Accelerator Virtualization and Isolation

Cloud environments benefit from sharing accelerators among multiple tenants [20, 28, 50, 62]. As FLD relies on PCIe peer-to-peer communication, a naive method of achieving this goal would be to implement SR-IOV on FLD, exposing multiple virtual functions and assigning each to its client VM. However, such an approach can cost additional hardware resources for implementing the additional virtualization logic in hardware and keep an additional per-function state. Instead, FLD exposes a single PCIe function and relies on NIC virtualization support (§ 5.4).

5 DESIGN

FLD's design allows direct interaction between accelerators and the NIC. FLD is embedded into the accelerator (on-die) and communicates with the NIC via peer-to-peer PCIe. The high-level design is shown in Figure 2d.

FLD forms a common hardware substrate, but the control plane software exposes different interfaces for different modes of operation. For accelerator disaggregation and inter-accelerator networking, we introduce the *FLD-R interface*, which utilizes the RDMA hardware transport implemented by the NIC. This interface allows accelerators to perform RDMA operations directly. For inline acceleration of network stack tasks, our *FLD-E* interface provides accelerators with a raw Ethernet interface, leveraging stateless offloads and NIC match-action rules.

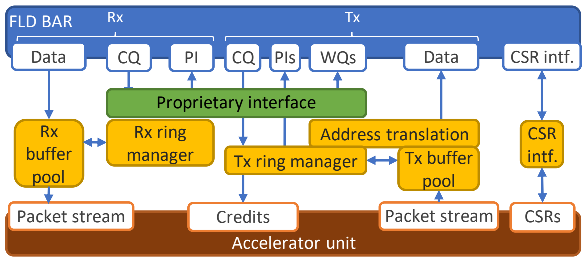


Figure 3: FLD hardware functional diagram.

5.1 Hardware Design

Figure 3 shows the high-level design of FLD hardware. FLD's address space, exposed over its PCIe BAR, is partitioned according to the various NIC data structures. A proprietary interface layer converts between the NIC's vendor-specific data structures and the FLD's internal formats.

Internally, FLD independently operates Tx and Rx modules. Both include buffer pools and a ring manager, which handles producer index accesses, descriptor accesses (for Tx), and completions writes. Ring managers maintain reference counts on their buffer pool and recycle buffers as needed.

The Tx side's address translation layer enables memory optimizations described in the next section.

5.2 FLD Memory Optimizations

The key observation for reducing FLD's memory requirements is that as a hardware device communicating over PCIe, *FLD does not need to store every queue in the exact format expected by the NIC.* Instead, it handles PCIe reads and writes with custom logic, generating the required data structures *on-the-fly* while keeping the data structures in a more space-optimized form under the hood. This opens up the opportunity for several optimizations that dramatically reduce the memory requirements compared to software.

Compression. The NIC descriptor and completion formats are more general than FLD's needs. For example, the FLD transmit queues always point to on-chip buffers, which are addressed with few bits, whereas the NIC interface accepts a 64-bit address. FLD

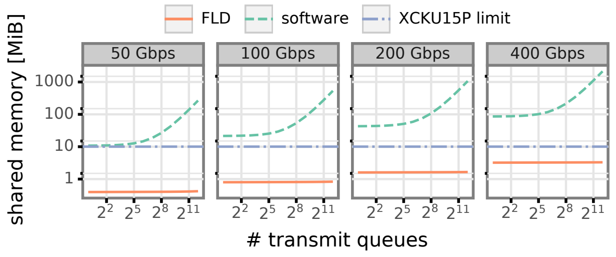


Figure 4: Driver memory requirements w./w.o. FLD optimizations. XCKU15P—on-chip memory on our prototype's FPGA.

internally uses a compressed descriptor/completion representation and converts them as needed.

Address Translation. Even with compression, scaling to a large number of queues would require a large Tx ring (S_{txq}, Table 3). To reduce it, we virtualize accesses to the descriptor ring region through an address translation mechanism. FLD maps PCIe reads from the queue's virtual address into a smaller physical array of descriptors. We use a 4-bank cuckoo hash-table (load factor $\frac{1}{2}$) to store a shared pool of N_{txdesc} descriptors. The hash table adds a 15.5 KiB overhead, but overall results in a 2080× reduction.

A similar translation table allows sharing transmit buffers belonging to different queues at fine granularity with bounded fragmentation. We map a per-queue virtual address range into the data buffer with another translation table taking 33 KiB, resulting in an overall 28.2× reduction.

Cuckoo hashing [86] provides constant-time lookup, resolving collisions by moving entries to a different location. In our implementation, when an inserted new entry collides, it evicts some old entry to a stash (containing four entries). The stash then tries to insert the evicted entry to another bank, and the process proceeds till success. If the stash fills up, insertion of a new entry stalls till some entry is released. To prevent backpressure, we double the table size, guaranteeing convergence.

MPRQ. We reduce receive-buffer fragmentation by leveraging suitable NIC mechanisms, such as multi-packet receive queues [26, 53] or packets spanning multiple buffers [43]. Specifically, the ConnectX-5 NIC we use in our prototype features multi-packet receive queues (MPRQs), receiving multiple packets in each buffer. MPRQs may still suffer from fragmentation but only up to half of the buffer size.

Receive Ring in Host Memory. We store the shared receive ring in host memory by designing FLD to recycle receive buffers in the same order initially posted. FLD can thus leave the descriptors unmodified.

Table 3 outlines the overall memory savings.

5.2.1 *Scalability Analysis.*

We analyze the memory scaling of both FLD and standard software for higher line-rates and numbers of transmit queues while using the rest of the parameters we used in the previous examples (Table 2a). Figure 4 shows that FLD scales well for 400 Gbps and 2048 transmit queues, whereas the memory requirements of the standard driver increase by orders of magnitude, clearly demonstrating the importance of the aforementioned memory optimizations.

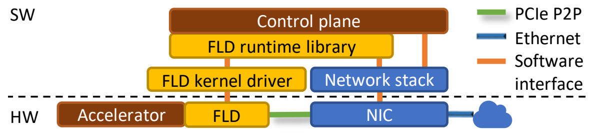


Figure 5: FLD high-level software design.

5.3 Software Control Plane

We show the main components of an FLD application in Figure 5. A control plane application running on the host CPU manages FLD through a shared FLD runtime library and a kernel driver to abstract the setup of FLD queues while utilizing the NIC's standard abstractions to configure how these queues operate.

FLD Runtime Library. FLD control-plane applications mainly use the runtime library to bind FLD and the NIC together. The application uses the library to create FLD queues (RDMA QPs or Ethernet transmit/receive queues) on behalf of the accelerator and uses the abstractions below to configure them.

The QP abstraction provided by the existing RDMA Verbs API serves as both a transport endpoint abstraction and an asynchronous I/O interface. It includes methods for, e.g., connecting a QP to a remote endpoint, but also for posting buffers for transmission. FLD-R QPs split these tasks: the accelerator uses it to transmit or receive data, while software only addresses its properties as a transport endpoint.

At a low level, the FLD runtime library allows the creation of Ethernet transmit and receive queues. Like the QP abstraction, such a queue can be associated with network flows through match-action rules as described above. However, this low-level abstraction complicates application development, as it requires developers to create additional rules to differentiate FLD packets from the remaining traffic explicitly.

FLD-E High-Level Abstraction. To simplify FLD-E applications, we extend the NIC's match-action APIs by adding new actions to those available today (§ 2.3). The new actions send packets to the accelerator along with appropriate metadata identifying the associated VM and the following table to process packets after acceleration. After processing, the accelerator returns the packet to the NIC, tagged with the next-table ID so that the NIC can resume processing the packet where the acceleration action took off.

FLD-R High-Level Abstraction. FLD-R applications can use our control plane as a standard RDMA server and have clients create RDMA connections that bind directly to FLD-R QPs.

Error Handling. As the CPU control plane manages NIC resources on behalf of the accelerator, it receives asynchronous error messages the NIC driver reports, similar to CPU applications. FLD hardware detects errors in the data plane and reports them to software through its kernel driver. Like RDMA Verbs, we leave error recovery to the control-plane application.

5.4 Virtualization and Isolation

Reusing the NIC I/O-virtualization features saves accelerator resources otherwise necessary to implement SR-IOV. To enable accelerator virtualization, the control plane configures the NIC to identify each message's tenant. FLD-R simply associates each queue with a single user, but for FLD-E, we allow more flexibility. Designers may prefer using queues to prioritize traffic while sharing a queue among multiple tenants.

To isolate and identify different flows, an FLD-E control-plane configures the NIC to tag ingress messages with a context ID associated with the tenant, based on their packet headers. FLD forwards this ID to the accelerator within the packet metadata. The accelerator also tags packets it sends, so the NIC can associate them with the appropriate tenant.

This design implies that untrusted VMs cannot control the context ID tag and require a trusted entity, e.g., the FLD-E control plane, to validate any match-action rules that they attempt to install.

5.5 FLD–Accelerator Interface

We design the interface between an accelerator and FLD around two AXI4-Stream buses, for receiving and transmitting packets. Such an interface is common for networking IP; for example, Xilinx Ethernet IP [123] supports a similar interface.

Although AXI4-Streams include flow control signals, we limit how accelerators may use them to bound the FLD buffer requirements and prevent the buffers from overflowing. When receiving data, the accelerator is not allowed to generate backpressure towards FLD, as that would eventually cause FLD buffers to fill up, and the NIC would drop incoming packets (for Ethernet) or stop traffic with transport-layer flow-control (for RDMA). Importantly, this would cause head-of-line blocking and may affect unrelated queues.

Instead, we expect accelerators to either meet line-rate, use application-layer flow-control to prevent incoming traffic from exceeding the accelerator receive throughput, or selectively drop exceeding traffic on their own.

When transmitting, each queue may progress at a different rate due to NIC prioritization (e.g., ETS) or transport-layer flow-/congestion-control. Therefore, we provide per-queue backpressure to the accelerator in the form of a credit interface. Accelerators may use it to monitor the per-queue tx data buffer and descriptor utilization and allocate available resources among the queues as they prefer.

Packets exchanged over the streaming buses are accompanied by metadata, such as the queue ID and context ID. Additionally, the metadata includes information derived from the completion notification the NIC provides with received packets, enabling the use of offloads such as checksum validation, packet parsing, classification, and RSS.

6 IMPLEMENTATION

We implement FLD as programmable logic on an NVIDIA Innova-2 Flex Open Programmable SmartNIC [78]. The Innova-2 includes both an NVIDIA ConnectX-5 NIC, and a Xilinx Kintex UltraScale+ FPGA (XCKU15P). The NIC embeds a PCIe switch and connects to the host and the FPGA through a PCIe Gen. 3 x8 link each (see Figure 6).

Table 4: Software lines of code [21] for different components.

Component	LOC	Component	LOC
FLD runtime library	3753	FLD-R control-plane	1510
FLD kernel driver	1137	FLD-R client library	754
FLD-E control-plane	1554	ZUC DPDK driver	732

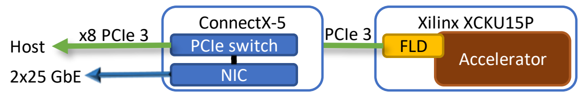


Figure 6: Innova-2 FPGA-based SmartNIC components.

As FLD relies on peer-to-peer PCIe, it is not limited to SmartNICs, but can also work with a separate NIC and FPGA boards connected through a PCIe switch or the host CPU's PCIe root complex. Nevertheless, we found optimizing for different PCIe fabrics difficult, as the performance depends on the PCIe fabric's latency. Bidirectional traffic can suffer degraded performance when control messages are delayed behind queued data messages. One possible solution is to tune switch buffers to match the latency the NIC expects, creating backpressure toward the NIC, and allowing the NIC to prioritize control messages as needed. This solution is platform-specific, however. Our choice of the Innova SmartNIC with its integrated PCIe switch simplified the task of using FLD in different servers.

We configure FLD to support two transmit queues. The receive/transmit buffers each have 256 KiB Transmit queues use a pool of 4096 descriptors. This configuration meets our PCIe and link bandwidth limitations and our testbed's latency. While the Innova-2 PCIe interface is limited to 50 Gbps, the FLD hardware interfaces operate at 100 Gbps.

We implement example control planes for FLD-R/E and a helper client library for accessing FLD-R control-plane remotely (see lines of code in Table 4).

PCIe Optimizations. To maximize PCIe utilization we use common optimizations such as selective completion signalling [54] and WQE-by-MMIO [55]. Multi-packet RQs (§ 4.3) also help due to their smaller descriptor number.

Limitations. Implementing a driver in hardware limits portability: a new NIC may have a different PCIe interface and require FLD modifications. Nevertheless, some NIC families have enough similarities to allow porting the design with minimal changes. For example, we have successfully tested our ConnectX-5-based design against ConnectX-6 Dx. In addition, some NICs offer standardized interfaces such as virtio [23, 82], and FlexDriver can be modified to support them. Thus, an accelerator using FlexDriver for a virtio-compatible NIC will work with any compliant NIC.

Our use of ConnectX-5's shared multi-packet RQ for RDMA messages improves latency. Messages comprising multiple packets generate completions when a packet arrives, even before the NIC receives the entire message. This allows processing the message incrementally, reducing latency [38], but requires the accelerator to handle the interleaving of packets of different queues.

The current FLD-E control-plane implementation uses static match-action rules instead of interacting with its applications.

7 APPLICATIONS

We develop three accelerators to highlight FLD's key benefits.

LTE Cipher Look-Aside Accelerator. We use FLD-R to develop a disaggregated ZUC cipher accelerator [67] (128-EEA3/EIA3 [106]). ZUC cipher is used in LTE mobile networks, where recent trends move processing to the cloud [87].

The disaggregated ZUC accelerator communicates with remote clients over RDMA Sends. Clients send cryptographic requests, and the accelerated server responds with an en/decrypted response. The request/response format includes a 64 B header for the cryptographic key, initialization vector (IV), and additional metadata. The ZUC accelerator comprises 8 ZUC modules, each operating, e.g., at 4.76 Gbps for 512 B messages, and a front-end load balancing unit.

We implement a client-side DPDK cryptodev driver for this accelerator using FLD-R interfaces. Compatibility with cryptodev APIs allows replacing an existing local accelerator (e.g., Intel QAT [42]) with our disaggregated one without software changes. The driver is easy to develop thanks to the use of the FLD-R client library (see LOC in Table 4).

IP Defragmentation. The IP protocol may fragment large packets into smaller frames to support networks with varying MTU sizes. IP fragmentation is commonly used for tunneling mobile [111] and multi-tenant traffic. Unfortunately, IP fragmentation prevents transport-layer NIC hardware offloads such as RSS and L4 checksums from operating correctly and increases CPU utilization, and techniques that attempt to prevent fragmentation may fail due to misconfiguration [64].

We implement an IP defragmentation offload as an example of a NIC packet processing extension, intended to intervene in the middle of the network stack to enable other NIC offloads after defragmenting IP packets.

We redirect fragmented packets to the accelerator at the embedded switch layer and use NIC offloads on the fragmented packets before and after defragmentation. In particular, we use VXLAN tunnel decapsulation offload before IP defragmentation. This is necessary due to pre-fragmentation, i.e., fragmenting packets before encapsulation or encryption to reduce the load on the decapsulating/decrypting endpoint [18, 36].

IP defragmentation could have also been implemented in a BITW design, but any pre-processing would have to be programmed into the FPGA, occupying some additional area. While a VXLAN decapsulation offload may not take much area, this is only one example. Cloud applications may require emerging hardware offloads such as IPSec [15, 41, 80], which are area-demanding [117]. FLD can use them transparently in the NIC, without reimplementing them in the accelerator.

IoT Token Authentication Offload. We implement a DDoS protection offload for IoT server applications [66]. The accelerator validates a cryptographic token provided within each message. The accelerator extracts a JSON Web Token [52] from CoAP-encoded messages [100], and validates the token, dropping packets with invalid HMAC-SHA256 signature. Our design supports 20 Mpps for 256 B packets using 8 processing units.

Table 5: Hardware: resource utilization and lines of code [21].

Module	Clk.	LUT	FF	BRAM	URAM	LOC
FLD	250	50K	66K	35	44	11K
PCIe core	250	12K	23K	44		
ZUC	200	38K	37K	242		6K
IP defrag.	250	17K	16K	984	64	2K
IoT auth.	200	118K	138K	293		8K

Unlike IP defragmentation, this offload serves user applications rather than accelerating the hypervisor's virtual switch. Thus, it shows how FLD-E can utilize NIC features for accelerator virtualization and performance isolation. Several tenants on the same host can share this offload, but each may have a different HMAC key. We rely on the NIC to identify flows that require authentication and tag them with the tenant identifier. The accelerator only needs a linear table of HMAC keys, indexed by the tag. In addition, sharing the accelerator among tenants requires QoS mechanisms to guarantee performance isolation. We use the traffic shaping capabilities of the NIC to implement maximum bandwidth shaping for the accelerator.

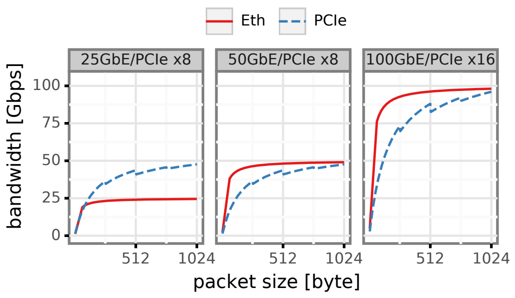
We chose this workload as it has already been implemented in previous work, NICA, on a BITW NIC [28], thus allowing for a direct comparison. In NICA, the design required a larger area (36% more LUTs, 40% more FFs, and 63% more BRAMs) while being 5.7× slower. The larger area footprint of NICA is partly because it reimplements some NIC features that FLD reuses. Specifically, NICA maps network flows to HMAC keys with its own flow table and QoS logic. As a BITW device, its NIC's ASIC processing only comes after the FPGA in the receive pipeline, preventing ASIC use.

8 EVALUATION

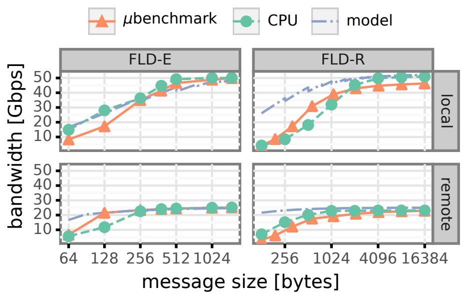
In our FLD evaluation, we aim to highlight the performance, area, and functional benefits of the FLD design, demonstrating that it occupies an advantageous point in the three-way trade-off that motivated this work (§ 3).

Setup. We run two kinds of experiments: local and remote. Local experiments stress FLD's PCIe interface by sending and receiving traffic from a host CPU to an accelerator, programmed on a local Innova-2 SmartNIC. The maximum throughput here is 50 Gbps limited by the PCIe. To run the FLD-E experiments, we associate one of the Innova-2 NIC's vPorts with the load generator, and another vPort with FLD-E, while the embedded switch is configured to loop back traffic between the two vPorts. In FLD-R experiments, we connect a client QP on the host to an FLD QP associated with the accelerator, where both QPs are associated with the same Innova-2 NIC.

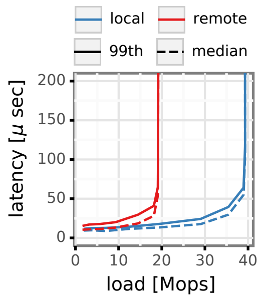
Remote experiments measure end-to-end network performance using a client node with a ConnectX-4 NIC and a server node with an Innova-2, connected back-to-back. We use 1500 B MTU for Ethernet and 1024 B for RoCE. The maximum throughput is 25 Gbps



(a) FLD performance model.



(b) FLD-R/E bandwidth vs. message size on Innova-2.



(c) FLD-R latency/bandwidth.

Figure 7: FLD microbenchmarks.

limited by the NIC's Ethernet interface. For FLD-E, we run the load generator on a client node and send traffic to the accelerator on a server node. Unless stated otherwise, we steer acceleration results to the software network stack to complete application processing. For FLD-R, we connect a QP on the client node to a remote FLD QP on the server node.

The client and the server run CentOS 7.0 on Haswell CPUs with 32 GiB of RAM.

8.1 FLD Characteristics

Performance Model. FLD communicates via PCIe, which implies a certain bandwidth overhead. Thus, it is unrealistic to expect line-rate throughput for any traffic pattern. To estimate an upper bound on the expected FLD performance that includes the PCIe overhead, we calculate the per-packet overhead and derive the expected throughput [27]. The overhead consists of control traffic associated with NIC–FLD communication, such as descriptors and completions.

PCIe vs. Raw Ethernet. Figure 7a compares the performance potential of FLD against a direct Ethernet link connected to the accelerator, as done in accelerator-hosted or BITW designs. We show the estimates for different network and PCIe rates. The 25 Gbps configuration to the left corresponds to our remote experiments. The model shows that the overheads allow meeting line rate of 25 Gbps for any packet size. The 50 Gbps configuration's PCIe line shows the expected performance for our local experiments. We can also conclude that FLD's current design can reach 95% of Ethernet line rate at 512 B packets for both 50 and 100 Gbps.

Some NIC optimizations for reducing PCIe traffic with small packets, such as inlining packet payload into the descriptor ring, compressing receive completions, and Enhanced Multi-Packet Write (EMPW) optimization [26] may further improve the performance for such traffic patterns. For a fair comparison, we disable EMPW in the CPU driver as well in FLD-E experiments below.

Area. We list the area utilized by FLD and its accompanying Xilinx PCIe core in Table 5, as well as the area of example accelerators. FLD area is comparable or smaller than the alternatives (Table 1), but it enables using all the NIC offloads, thus providing a much richer set of features than the competitors.

Table 6: Network echo round-trip for 64 B packets in μs.

	Mean	Median	99th-%	99.9th-%
FLD-E	2.78	2.6	3.4	4.34
CPU	2.36	2.34	2.58	11.18
Emu [107]	1.09		1.11	

8.1.1 FLD-E. To evaluate FLD-E raw performance, we use a simple echo FLD-E accelerator, which sends back each packet it receives. The goal is to estimate the overhead of using FLD-E vs. a regular CPU driver. We configure the Innova-2 NIC to send FLD-E outbound traffic back to the client. We use the DPDK testpmd application to generate packets towards the accelerator while measuring the bandwidth and the latency and compare it against a testpmd implementation on the CPU and our performance model's estimate.

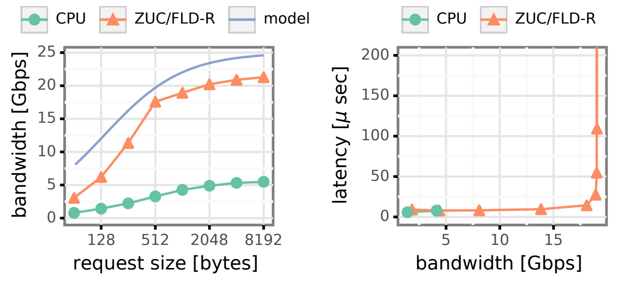
Figure 7b shows the bandwidth for different packet sizes. We see that FLD meets the expected performance for Ethernet packets starting with 128 B and 256 B for the remote and local interfaces, respectively, and its performance is on par with a CPU driver.

We further measure the throughput of the FLD-E echo accelerator while forwarding packets of mixed sizes taken from the IMC 2010 data-center trace [9]. FLD-E is able to process 12.7 Mpps, compared to 9.6 Mpps on a single CPU core with DPDK testpmd, showing that FLD can drive the NIC as efficiently as the CPU.

We test FLD-E latency on an empty system with the same setup, comparing it to an echo server running on the CPU. Table 6 shows that FLD-E increases the mean latency by 17 % compared to the CPU, likely due to the slower FPGA clock rate. However, it improves 99.9th percentile latency by 2.5× because there is no OS interference with the network stack.

Compared to a direct-attached FPGA or a BITW design, where the accelerator can respond directly to the network, FLD increases latency as packets go through the ASIC NIC and a PCIe link in every direction. For example, our latency is 1.69 μs higher than [107]). Nevertheless, we believe that a slight latency increase might often be worth the exchange for the ability to use additional features provided by the NIC.

8.1.2 FLD-R. We measure FLD-R throughput by using the echo accelerator and connecting it to RDMA QPs. The results are in Figure 7b's right column. While slightly lower than FLD-E, FLD-R



(a) Bandwidth vs. request size. **(b) Latency vs. bandwidth.**

Figure 8: Disaggregated ZUC accelerator performance.

remote performance meets its 25 Gbps target line rate for messages larger than 512 B. For smaller packets, the system is bottlenecked by the CPU client. Unfortunately, we did not find the reason for the reduced performance in local FLD-R experiments. The experiment also demonstrates how FLD-R uses messages larger than the MTU, relying on the NIC's transport for segmenting packets in hardware.

Figure 7c shows the 1 KiB messages latency vs. throughput while varying the load. At a low load, the median latency is 9.4 µs for local access and 10.6 µs for remote. As the load increases, queuing delay dominates the latency; FLD reaches a bottleneck near 82 % of the expected local/remote bandwidth.

8.2 End-to-End Applications

8.2.1 Disaggregated LTE Cipher Accelerator. We measure the performance of the ZUC accelerator using DPDK's test-crypto-perf [110] on the client. Figure 8a compares the encryption throughput when accessing the remote accelerator (25 Gbps link) against a local CPU implementation. We also show the upper bound estimate produced by the performance model, which considers RoCE headers and the 64B app headers accompanying each request/response. The CPU implementation uses DPDK's software ZUC driver, based on Intel Multi-Buffer Crypto Library [44]. For request size ≥ 512 B FLD reaches 17.6 Gbps, 89 % of the expected bandwidth, and 4× higher than the CPU throughput. This result can be further improved by adding on-FPGA key storage and request batching, which we leave to future work.

Figure 8b shows latency vs. bandwidth graphs. The disaggregated accelerator is not faster at low loads, but it is still worth using to pool resources and free the CPU core consumed by the software implementation.

Overall, the ZUC experiments show how FLD promotes the development of high-performance disaggregated accelerators.

8.2.2 IP Defragmentation. We compare our defragmentation offload against a baseline that decapsulates VXLAN traffic using NIC offloads and then defragments packets in software. We measure throughput using 60 iperf TCP flows and compare three configurations:

(a) no fragmentation,
(b) 1500 B packets fragmented over a route configured with a 1450 B MTU (without encapsulation), and
(c) 1500 B packets fragmented and sent over a VXLAN tunnel with a 1450 B MTU.

Several NIC offloads break on fragmented packets, including RSS and checksum, and consequently, performance drops from 23.2 Gbps to 3.2 Gbps with software defragmentation. Without RSS, most packets default to a single receiver-core, which becomes the bottleneck. Hardware defragmentation enables RSS and reaches 22.4 Gbps, a 7× speedup. With VXLAN decapsulation, we observe a lower speedup (5.25× compared to the CPU-only version) because the sender becomes the bottleneck, as in our setup, it relies on software fragmentation and tunneling.

This experiment demonstrates that FLD enables injecting the acceleration in the middle of the packet processing pipeline, enabling NIC packet processing offloads to be invoked both before and after the accelerator.

8.2.3 IoT Token Authentication Offload. We test the accelerator in the remote setting by generating varying request sizes using TRex [114]. We observe the offload meets line-rate for packets ≥ 256 B (not shown).

We also validate that the accelerator indeed provides performance isolation among different tenants by sending two competing flows to the receiver, with 8 Gbps and 16 Gbps respectively. Here the accelerator is configured to accept only 12 Gbps of traffic. Without shaping, the accelerator admits flow traffic in proportion to the flow's link utilization, resulting in an uneven allocation of the accelerator among the tenants (4.15 Gbps vs. 8.35 Gbps). By setting a 6 Gbps limit to both flows, flow B cannot exceed its limit and thus allows flow A to reach its allocated bandwidth.

Overall, this experiment demonstrates the ability of FLD to enable accelerator virtualization by relying on the NIC for flow marking, virtualization, and traffic shaping.

9 DISCUSSION

Even though our Innova-2 implementation of FLD is limited to 50 Gbps, we believe FLD can scale to higher speeds. We identify two primary potential obstacles to scaling: internal fabric (PCIe) bandwidth and FlexDriver design scalability. For the former, we believe that internal fabric speeds will increase proportionally to future network speeds. Thus, a 400 Gbps NIC should communicate at 400 Gbps with an accelerator through a future fabric such as PCIe 5.0 or CXL. As for the latter, the current FLD implementation is clocked to process up to 100Gbps. We believe the design can scale either by increasing the pipeline width or instantiating multiple FLD "cores" within the accelerator, combined with NIC RSS offloads to balance the load on these cores. Higher bandwidth will also require larger buffers, but they are still within reasonable memory capacity bounds (§ 5.2.1).

10 RELATED WORK

Our work builds upon previous GPU networking work to use NICs directly from GPUs, eliminating the CPU from the critical path [2, 22, 60, 84, 91]. These works implement communication tasks in GPGPU cores or CUDA stream MemOps [2]. In contrast, we implement FLD as a hardware module, suitable even for accelerators that lack GPGPU-like programmability.

FlexNIC [56] suggests using match-action programmable DMA to offload GPU communication tasks. Lynx [113] uses SmartNIC ARM cores to execute network stack functionality and enable accelerator networking. Both adapt the NIC's DMA interfaces to simplify accelerator logic. Contrarily, we embed FlexDriver near the accelerator, supporting NICs without programmable DMA interfaces or ARM cores.

New fabrics designed for disaggregation such as Gen-Z [35] and CXL [118] can connect remote accelerators and their clients. However, these are designed for rack-/row-scale, and accelerators may still require NICs for data-center-wide traffic.

We expand upon the alternative techniques described in § 3.

CPU-Mediated. Early attempts at accelerator networking relied on CPUs to orchestrate GPU networking [57, 90, 92]. CPU performance limits such an approach.

Accelerator-Hosted. NetFPGA [125] and other FPGA NICs [33, 103] enable prototyping new kinds of networking offloads [85], architectures [13, 62, 89, 120], and applications [8, 112]. Similar applications can be prototyped over FlexDriver, relying on external NICs instead.

Previous work suggested using FPGAs as accelerators directly attached to the network or utilizing FPGA-based SmartNICs this way. NMU [96] provides anti-spoofing and tunneling for direct-connected FPGAs. Microsoft's Catapult [14] implements its LTL transport on FPGAs, serving disaggregated [17] and distributed [93] applications. Sidler et. al has implemented network stacks for TCP/IP [101, 102] and RoCE [103] for FPGA applications, used by frameworks such as Galapagos [30], NICA [28], Coyote [58], and IBM's network-attached FPGAs [121]. We try to offer similar networking services for accelerators using an external ASIC NIC.

Habana Gaudi DNN accelerator embeds a RoCE v2 NIC [68]. The integrated NIC lacks several features of a full-scale NIC, such as virtualization and memory protection [34]. Others use a dedicated fabric for accelerator–accelerator communication [75, 83, 93]. Our goal is to connect accelerators to a converged data-center network.

Bump-in-the-wire NICs [14, 45, 69] use an ASIC NIC for some of its functionality. Using the NIC's DMA interfaces for SR-IOV and RDMA, AccelNet implements SDN in FPGA [32], and NICA implements inline application acceleration [28]. In contrast, FLD enables additional NIC offloads.

FPGA Network Programming Frameworks. Programming languages and hardware development frameworks such as Emu [107], P4 [119, 124] and implementations of Click for FPGAs [61, 95] simplify building packet processing accelerators. FlexDriver complements these by offloading some of their functionality to the NIC.

NIC Offloads. In addition to those in § 2. NIC vendors offer a variety of offloads, e.g., for cryptography [15, 41, 80, 88] and storage [11, 41, 63]. Some offer full TCP offload engines [72]. Amazon EFA offloads reliable datagrams [99], and 1RMA [104] offers partial transport offloads for flow control and security. Our design provides accelerators with access to such offloads.

11 CONCLUSION

Efficient accelerator communication with the network is essential for building disaggregated, distributed, or inline accelerators. FlexDriver offers a new method for connecting accelerators, fully utilizing existing NIC functionality, with minimal CPU overhead and area utilization. Accelerators drive the NIC over peer-to-peer PCIe fabric and benefit from NIC offloads such as RDMA, tunneling, and traffic shaping.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Jing Li for their insightful comments. Our work is built upon the contributions of NVIDIA Networking architecture, software development, and chip-design teams, notably Eitan Ziv, Igor Leshchenko, Meir Lichtinger, Ran Koren, and Vadim Artishchev. This work was partially supported by the Technion Hiroshi Fujiwara Cyber Security Research Center, the Israel National Cyber Directorate, and the Israel Innovation Authority. We gratefully acknowledge support from Israel Science Foundation (Grant 1027/18).

REFERENCES

[1] Francois Abel, Jagath Weerasinghe, Christoph Hagleitner, Beat Weiss, and Stephan Paredes. 2017. An FPGA Platform for Hyperscalers. In *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*. 29–32. https://doi.org/10.1109/HOTI.2017.13

[2] Elena Agostini, Davide Rossetti, and Sreeram Potluri. 2018. GPUDirect Async: Exploring GPU synchronous communication techniques for InfiniBand clusters. *J. Parallel and Distrib. Comput.* 114 (2018), 28–45. https://doi.org/10.1016/j.jpdc.2017.12.007

[3] Amazon Web Services. 2016. Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/ (Accessed: Nov. 2021).

[4] Amazon Web Services. 2019. AWS Nitro System. https://aws.amazon.com/ec2/nitro/ (Accessed: Aug. 2021).

[5] Jeff Barr. 2010. New EC2 Instance Type – The Cluster GPU Instance. https://aws.amazon.com/blogs/aws/new-ec2-instance-type-the-cluster-gpu-instance/ (Accessed: Dec. 2021).

[6] Stephen Bates. 2015. Donard: NVM Express for Peer-2-Peer between SSDs and other PCIe Devices *(SDC 2015)*. SNIA, Santa Clara, CA, USA. https://www.snia.org/sites/default/files/SDC15_presentations/nvme_fab/StephenBates_Donard_NVM_Express_Peer-2_Peer.pdf.

[7] Stephen Bates. 2018. Avoiding the NVM Express bottleneck with NVMe CMBs, Eideticom and SPDK. https://www.eideticom.com/media-news/blog/25-avoiding-the-nvm-express-bottleneck-with-nvme-cmbs-eideticom-and-spdk.html (Accessed: Aug. 2021).

[8] Giacomo Belocchi, Valeria Cardellini, Aniello Cammarano, and Giuseppe Bianchi. 2020. Paxos in the NIC: Hardware Acceleration of Distributed Consensus Protocols. *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020* (2020), 1–6.

[9] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (Melbourne, Australia) *(IMC '10)*. Association for Computing Machinery, New York, NY, USA, 267–280. https://doi.org/10.1145/1879141.1879175

[10] Shai Bergman, Tanya Brokhman, Tzachi Cohen, and Mark Silberstein. 2019. SPIN: Seamless Operating System Integration of Peer-to-Peer DMA Between SSDs and GPUs. *ACM Trans. Comput. Syst.* 36, 2, Article 5 (April 2019), 26 pages. https://doi.org/10.1145/3309987

[11] Broadcom. 2019. NVMe over Fabrics Performance Stingray™-Based Storage Appliance. https://docs.broadcom.com/doc/broadcom-stingray-100G-NVMe-oF-performance (Accessed: Dec. 2021).

[12] Broadcom. 2020. NetXtreme E-Series PCIe NIC Ethernet Adapters Specification Sheet. https://docs.broadcom.com/doc/NetXtreme-E-PCIENIC-SG (Accessed: Aug. 2021).

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

[14] Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A Cloud-Scale Acceleration Architecture. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-49)*. IEEE Computer Society. https://www.microsoft.com/en-us/research/publication/configurable-cloud-acceleration/

[15] Chelsio Communications. 2018. Accelerated IPsec-VPN Communication with T6. https://www.chelsio.com/wp-content/uploads/resources/t6-100g-ipsec-linux.pdf (Accessed: Jul. 2021).

[16] Jongsok Choi, Ruolong Lian, Zhi Li, Andrew Canis, and Jason Anderson. 2018. Accelerating Memcached on AWS Cloud FPGAs. In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies* (Toronto, ON, Canada) *(HEART 2018)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/3241793.3241795

[17] Eric S. Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian M. Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Husseini, Tamás Juhász, Kara Kagi, Ratna Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven K. Reinhardt, Bita Darvish Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. 2018. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro* 38, 2 (March 2018), 8–20. https://doi.org/10.1109/MM.2018.022071131

[18] Cisco. 2020. Pre-Fragmentation for IPsec VPNs. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_conn_dplane/configuration/xe-16-12/sec-ipsec-data-plane-xe-16-12-book/sec-pre-frag-vpns.pdf (Accessed: May. 2021).

[19] NVIDIA Corporation. 2018. GPU-Accelerated Microsoft Azure. https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/microsoft-azure/ (Accessed: Jul. 2021).

[20] NVIDIA Corporation. 2020. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/ (Accessed: Aug. 2021).

[21] Al Danial. 2021. cloc–count lines of code. https://github.com/AlDanial/cloc (Accessed: Jul. 2021).

[22] Feras Daoud, Amir Watad, and Mark Silberstein. 2016. GPUrdma: GPU-side Library for High Performance Networking from GPU Kernels. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers* (Kyoto, Japan) *(ROSS '16)*. ACM, New York, NY, USA, Article 6, 8 pages. https://doi.org/10.1145/2931088.2931091

[23] Sujal Das and Abhijeet Prabhune. 2016. Netronome – The Case for Express Virtio (XVIO). https://www.netronome.com/blog/the-case-for-express-virtio-xvio-part-1/ (Accessed: Nov. 2021).

[24] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan. 2012. High performance network virtualization with SR-IOV. *J. Parallel and Distrib. Comput.* 72, 11 (2012), 1471–1480.

[25] DPDK. 2018. Generic flow API (rte_flow). https://doc.dpdk.org/guides/prog_guide/rte_flow.html (Accessed: Jul. 2021).

[26] DPDK. 2019. MLX5 poll mode driver. https://doc.dpdk.org/guides/nics/mlx5.html (Accessed: Jul. 2021).

[27] Haggai Eran. 2021. FlexDriver performance and memory utilization models. https://github.com/acsl-technion/flexdriver-model

[28] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. 2019. NICA: An Infrastructure for Inline Acceleration of Network Applications. In *2019 USENIX Annual Technical Conference (USENIX ATC 19) (USENIX ATC 2019)*. USENIX Association, Renton, WA, 345–362. https://www.usenix.org/conference/atc19/presentation/eran

[29] Mesut Ergin, Harry Van Haaren, and Charlie Tai. 2019. Intel Ethernet Controller 700 Series - Open vSwitch Hardware Acceleration Application Note. https://builders.intel.com/docs/networkbuilders/intel-ethernet-controller-700-series-open-vswitch-hardware-acceleration-application-note.pdf

[30] Nariman Eskandari, Naif Tarafdar, Daniel Ly-Ma, and Paul Chow. 2019. A Modular Heterogeneous Stack for Deploying FPGAs and CPUs in the Data Center. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 262–271. https://doi.org/10.1145/3289602.3293909

[31] Alireza Farshin, Amir Roozbeh, Gerald Q. Maguire Jr., and Dejan Kostić. 2020. Reexamining Direct Cache Access to Optimize I/O Intensive Applications for Multi-hundred-gigabit Networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 673–689. https://www.usenix.org/conference/atc20/presentation/farshin

[32] 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) (NSDI '18)*. USENIX Association, Renton, WA, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[33] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps NIC. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 38–46. https://doi.org/10.1109/FCCM48280.2020.00015

[34] Oded Gabbay. 2020. Adding GAUDI NIC code to habanalabs driver. https://lwn.net/Articles/831227/ (Accessed: Aug. 2021).

[35] Gen-Z Consortium. 2016. Gen-Z Overview. https://genzconsortium.org/wp-content/uploads/2018/05/Gen-Z-Overview-V1.pdf (Accessed: Jul. 2021).

[36] Google Cloud. 2020. Google Cloud VPN documentation – MTU considerations. https://cloud.google.com/network-connectivity/docs/vpn/concepts/mtu-considerations (Accessed: Dec. 2021).

[37] Anubhav Guleria, J Lakshmi, and Chakri Padala. 2019. QuADD: QUantifying Accelerator Disaggregated Datacenter Efficiency. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 349–357. https://doi.org/10.1109/CLOUD.2019.00064

[38] Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing in the Network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC17) (SC 2017)*.

[39] Bert Hubert, Thomas Graf, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, and Pedro Larroy. 2012. Linux Advanced Routing & Traffic Control HOWTO. https://lartc.org/howto/

[40] InfiniBand Trade Association. 2004. In *InfiniBand Architecture Specification – Release 1.2*. Vol. 1. Chapter 10.2.9 Shared Receive Queue.

[41] Intel. 2013. Intel Ethernet Converged Network Adapter X540. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-x540-t2-brief.pdf (Accessed: Jul. 2021).

[42] Intel. 2015. Intel QuickAssist Adapter 8950 Product Brief. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/quickassist-adapter-8950-brief.pdf. Accessed: Jun. 2021.

[43] Intel. 2019. Intel 82599 10 GbE Controller Datasheet. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82599-10-gbe-controller-datasheet.pdf (Accessed: Aug. 2021).

[44] Intel. 2019. Intel® Multi-Buffer Crypto for IPsec Library. https://github.com/intel/intel-ipsec-mb (Accessed: Jul. 2021).

[45] Intel. 2020. Intel FPGA Programmable Acceleration Card N3000. https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/intel-fpga-pac-n3000/overview.html (Accessed: Jul. 2021).

[46] Intel. 2020. Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA. https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/acceleration-card-arria-10-gx/overview.html (Accessed: Jul. 2021).

[47] Intel. 2020. Intel Scalable I/O Virtualization Technical Specifications. https://software.intel.com/content/www/us/en/develop/download/intel-scalable-io-virtualization-technical-specification.html (Accessed: Jun. 2021).

[48] Zsolt István, David Sidler, and Gustavo Alonso. 2017. Caribou: Intelligent Distributed Storage. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1202–1213. https://doi.org/10.14778/3137628.3137632

[49] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. 2016. Consensus in a Box: Inexpensive Coordination in Hardware. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 425–438. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/istvan

[50] Zsolt István, Gustavo Alonso, and Ankit Singla. 2018. Providing Multi-tenant Services with FPGAs: Case Study on a Key-Value Store. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL) (FPL 2018)*. 119–1195. https://doi.org/10.1109/FPL.2018.00029

[51] Mike Jackson and Ravi Budruk. 2012. *PCI Express Technology: Comprehensive Guide to Generations 1.x, 2.x, 3.0.* MindShare, Inc.

[52] Michael Jones, John Bradley, and Nat Sakimura. 2015. JSON Web Token (JWT). RFC 7519. https://doi.org/10.17487/RFC7519

[53] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 1–16. https://www.usenix.org/conference/nsdi19/presentation/kalia

[54] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA Efficiently for Key-Value Services. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 295–306. https://doi.org/10.1145/2619239.

2626299

[55] 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

[56] Antoine Kaufmann, Simon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 67–81. https://doi.org/10.1145/2872362.2872367

[57] Sangman Kim, Seonggu Huh, Xinya Zhang, Yige Hu, Amir Wated, Emmett Witchel, and Mark Silberstein. 2014. GPUnet: Networking Abstractions for GPU Programs. In *11th USENIX Symposium on Operating System Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 201–216. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/kim

[58] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs?. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe

[59] Patrick Kutch. 2011. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf Intel application note 321211–002.

[60] Michael LeBeane, Khaled Hamidouche, Brad Benton, Mauricio Breternitz, Steven K. Reinhardt, and Lizy K. John. 2017. GPU Triggered Networking for Intra-kernel Communications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. ACM, New York, NY, USA, Article 22, 12 pages. https://doi.org/10.1145/3126908.3126950

[61] Bojie Li, Kun Tan, Layong Larry Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/2934872.2934897

[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] Liran Liss. 2018. NVMf Target Offload. https://www.openfabrics.org/images/2018workshop/presentations/308_LLiss_NVMfTargetOffload.pdf (Accessed: Aug. 2021).

[64] Matthew Luckie, Kenjiro Cho, and Bill Owens. 2005. Inferring and Debugging Path MTU Discovery Failures. In *Internet Measurement Conference 2005 (IMC 05)*. USENIX Association, Berkeley, CA. https://www.usenix.org/conference/imc-05/inferring-and-debugging-path-mtu-discovery-failures

[65] Layong Larry Luo. 2018. Towards Converged SmartNIC Architecture for Bare Metal & Public Clouds. https://conferences.sigcomm.org/events/apnet2018/slides/larry.pdf APNet 2018 (Accessed: Aug. 2021).

[66] Gabi Malka. 2021. FlexDriver IoT authentication offload example AFU. https://github.com/acsl-technion/flexdriver-iot-auth

[67] Gabi Malka. 2021. FlexDriver ZUC cipher example AFU. https://github.com/acsl-technion/flexdriver-zuc

[68] Eitan Medina and Eran Dagan. 2020. Habana Labs Purpose-Built AI Inference and Training Processor Architectures: Scaling AI Training Systems Using Standard Ethernet With Gaudi Processor. *IEEE Micro* 40, 2 (2020), 17–24. https://doi.org/10.1109/MM.2020.2975185

[69] Mellanox Technologies. 2017. Innova™ Flex 4 Lx EN Adapter Card Product Brief. https://www.mellanox.com/related-docs/prod_adapter_cards/PB_Innova_Flex4_Lx_EN.pdf (Accessed: Dec. 2021).

[70] Mellanox Technologies. 2019. Mellanox ASAP^2 : Accelerated Switching and Packet Processing. https://www.mellanox.com/files/doc-2020/sb-asap2.pdf (Accessed: Dec. 2021).

[71] Microsoft Azure. 2021. New NPv1 virtual machines are now generally available. https://azure.microsoft.com/en-us/updates/new-npv1-virtual-machines-are-now-generally-available/ (Accessed: Jul. 2021).

[72] Jeffrey C. Mogul. 2003. TCP Offload is a Dumb Idea Whose Time Has Come. In *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9* (Lihue, Hawaii) *(HOTOS'03)*. USENIX Association, Berkeley, CA, USA, 5–5. http://dl.acm.org/citation.cfm?id=1251054.1251059

[73] napa:tech;. 2021. Link Programmable. https://www.napatech.com/products/link-programmable/ (Accessed: Jun. 2021).

[74] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. 2018. Understanding PCIe Performance for End Host Networking. 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, 327–341. https://doi.org/10.1145/3230543.3230560

[75] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman Jouppi, and David Patterson. 2021. The Design Process for Google's Training Chips: TPUv2 and TPUv3. *IEEE Micro* 41, 2 (2021), 56–63. https://doi.org/10.1109/MM.2021.3058217

[76] NVIDIA Corporation. 2011. NVIDIA GPUDirect. https://developer.nvidia.com/gpudirect (Accessed: Aug. 2021).

[77] NVIDIA Corporation. 2019. GPUDirect Storage: A Direct Path Between Storage and GPU Memory. https://developer.nvidia.com/blog/gpudirect-storage/ (Accessed: Aug. 2021).

[78] NVIDIA Corporation. 2020. Innova™-2 Flex. https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/ (Accessed: Aug. 2021).

[79] NVIDIA Corporation. 2020. Networking Product Documentation – MLNX_OFED v5.0-2.1.8.0 – Mediated Devices. https://docs.mellanox.com/display/OFEDv502180/Mediated+Devices (Accessed: Dec. 2021).

[80] NVIDIA Corporation. 2021. ConnectX®-6 Dx Ethernet SmartNIC. https://www.mellanox.com/files/doc-2020/pb-connectx-6-dx-en-card.pdf (Accessed: Jul. 2021).

[81] NVIDIA Corporation. 2021. GPUDirect RDMA: Synchronization and Memory Ordering. https://docs.nvidia.com/cuda/gpudirect-rdma/#sync-behavior (Accessed: Aug. 2021).

[82] NVIDIA Corporation. 2021. NVIDIA BlueField®-2 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf (Accessed: Nov. 2021).

[83] NVIDIA Corporation. 2021. NVLink and NVSwitch. https://www.nvidia.com/en-us/data-center/nvlink/ (Accessed: Aug. 2021).

[84] Lena Oden, Holger Fröning, and Franz-Joseph Pfreundt. 2014. Infiniband-Verbs on GPU: A Case Study of Controlling an Infiniband Network Device from the GPU. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*. 976–983. https://doi.org/10.1109/IPDPSW.2014.111

[85] Racyus Pacifico, Matheus S. Castanho, L. Vieira, M. Vieira, Lucas F. S. Duarte, and J. Nacif. 2021. Application Layer Packet Classifier in Hardware. *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2021), 515–522.

[86] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144. https://doi.org/10.1016/j.jalgor.2003.12.002

[87] Mugen Peng, Yaohua Sun, Xuelong Li, Zhendong Mao, and Chonggang Wang. 2016. Recent Advances in Cloud Radio Access Networks: System Architectures, Key Techniques, and Open Issues. *IEEE Communications Surveys Tutorials* 18, 3 (2016), 2282–2308. https://doi.org/10.1109/COMST.2016.2548658

[88] Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafrir. 2021. Autonomous NIC Offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 18–35. https://doi.org/10.1145/3445814.3446732

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

[90] Sreeram Potluri, Devendar Bureddy, Khaled Hamidouche, Akshay Venkatesh, Krishna Kandalla, Hari Subramoni, and Dhabaleswar K. Panda. 2013. MVAPICH-PRISM: A proxy-based communication framework using InfiniBand and SCIF for Intel MIC clusters. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–11. https://doi.org/10.1145/2503210.2503288

[91] Sreeram Potluri, Anshuman Goswami, Davide Rossetti, Chris J. Newburn, Manjunath Gorentla Venkata, and Neena Imam. 2017. GPU-Centric Communication on NVIDIA GPU Clusters with InfiniBand: A Case Study with OpenSHMEM. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. 253–262. https://doi.org/10.1109/HiPC.2017.00037

[92] Sreeram Potluri, Khaled Hamidouche, Akshay Venkatesh, Devendar Bureddy, and Dhabaleswar K. Panda. 2013. Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs. In *2013 42nd International Conference on Parallel Processing*. 80–89. https://doi.org/10.1109/ICPP.2013.17

[93] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2016. A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services. *Commun. ACM* 59, 11 (Oct. 2016), 114–122. https://doi.org/10.1145/2996868

[94] Yong Ren. 2018. High performance Cloud with Hardware Acceleration. https://conferences.sigcomm.org/events/apnet2018/slides/yong.pdf APNet 2018 (Accessed: Aug. 2021).

[95] Teemu Rinta-aho, Mika Karlstedt, and Madhav P. Desai. 2012. The Click2NetFPGA Toolchain. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX, Boston, MA, 77–88. https://www.usenix.org/conference/atc12/technical-sessions/presentation/rinta-aho

[96] Daniel Rozhko and Paul Chow. 2019. The Network Management Unit (NMU): Securing Network Access for Direct-Connected FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 232–241. https://doi.org/10.1145/3289602.3293903

[97] Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso, and Sergio López-Buedo. 2019. Limago: An FPGA-Based Open-Source 100 GbE TCP/IP Stack. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 286–292. https://doi.org/10.1109/FPL.2019.00053

[98] Simon Sarwood. 2017. Amazon reveals 'Nitro'... Custom ASICs and boxes that do grunt work so EC2 hosts can just run instances. https://www.theregister.com/2017/11/29/aws_reveals_nitro_architecture_bare_metal_ec2_guard_duty_security_tool/ (Accessed: Aug. 2021).

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

[100] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. The Constrained Application Protocol (CoAP). RFC 7252. https://doi.org/10.17487/RFC7252

[101] David Sidler, Gustavo Alonso, Michaela Blott, Kimon Karras, Kees Vissers, and Raymond Carley. 2015. Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. 36–43. https://doi.org/10.1109/FCCM.2015.12

[102] David Sidler, Zsolt István, and Gustavo Alonso. 2016. Low-latency TCP/IP stack for data center applications. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL) (FPL 2016)*. 1–4. https://doi.org/10.1109/FPL.2016.7577319

[103] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: Smart Remote Memory. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. https://doi.org/10.1145/3342195.3387519

[104] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-Envisioning Remote Memory Access for Multi-Tenant Datacenters. 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, 708–721. https://doi.org/10.1145/3387514.3405897

[105] Igor Smolyar, Alex Markuze, Boris Pismenny, Haggai Eran, Gerd Zellweger, Austin Bolen, Liran Liss, Adam Morrison, and Dan Tsafrir. 2020. IOctopus: Outsmarting Nonuniform DMA. 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, 101–115. https://doi.org/10.1145/3373376.3378509

[106] ETSI/SAGE Specification. 2011. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification. https://www.gsma.com/aboutus/wp-content/uploads/2014/12/eea3eia3zucv16.pdf Version 1.6.

[107] Nik Sultana, Salvator Galea, David Greaves, Marcin Wojcik, Jonny Shipton, Richard Clegg, Luo Mai, Pietro Bressana, Robert Soulé, Richard Mortier, Paolo Costa, Peter Pietzuch, Jon Crowcroft, Andrew W Moore, and Noa Zilberman. 2017. Emu: Rapid Prototyping of Networking Services. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 459–471. https://www.usenix.org/conference/atc17/technical-sessions/presentation/sultana

[108] S. Sur, Lei Chai, Hyun-Wook Jin, and D.K. Panda. 2006. Shared receive queue based scalable MPI design for InfiniBand clusters. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. https://doi.org/10.1109/IPDPS.2006.1639336

[109] The kernel development community. 2019. Linux Networking Documentation. https://www.kernel.org/doc/html/latest/networking/index.html (Accessed: Aug. 2021).

[110] The Linux Foundation Projects. 2018. DPDK Tools User Guides – dpdk-test-crypto-perf Application. https://doc.dpdk.org/guides/tools/cryptoperf.html (Accessed: Jul. 2021).

[111] Juha-Matti Tilli and Raimo Kantola. 2017. Data plane protocols and fragmentation for 5G. In *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*. 207–213. https://doi.org/10.1109/CSCN.2017.8088623

[112] Yuta Tokusashi, Hiroki Matsutani, and Noa Zilberman. 2018. LaKe: The Power of In-Network Computing. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig) (ReConFig'18)*. 1–8. https://doi.org/10.1109/RECONFIG.2018.8641696

[113] Maroun Tork, Lina Maudlej, and Mark Silberstein. 2020. Lynx: A SmartNIC-Driven Accelerator-Centric Architecture for Network Servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 117–131. https://doi.org/10.1145/3373376.3378528

[114] TRex Team. 2016. TRex: Realistic Traffic Generator. https://trex-tgn.cisco.com/ (Accessed: Aug. 2021).

[115] Animesh Trivedi. 2011. Remote Direct Memory Access (RDMA) 101 – Quick History Lesson and Introduction. http://0x8086.blogspot.com/2011/11/remote-direct-memory-access-rdma-101.html (Accessed: Aug. 2021).

[116] Vojislav Đukić, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, and Ankit Singla. 2019. Is advance knowledge of flow sizes a plausible assumption?. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 565–580. https://www.usenix.org/conference/nsdi19/presentation/dukic

[117] Markku Vajaranta, Arto Oinonen, Timo D. Hämäläinen, Vili Viitamäki, Jouni Markunmäki, and Ari Kulmala. 2019. Feasibility of FPGA accelerated IPsec on cloud. *Microprocessors and Microsystems* 71 (2019), 102861. https://doi.org/10.1016/j.micpro.2019.102861

[118] S. Van Doren. 2019. Abstract - HOTI 2019: Compute Express Link. In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*. 18–18. https://doi.org/10.1109/HOTI.2019.00017

[119] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. 2017. P4FPGA: A Rapid Prototyping Framework for P4. In *Proceedings of the Symposium on SDN Research (SOSR 2017)*. ACM, 122–135.

[120] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. 2021. Isolation mechanisms for high-speed packet-processing pipelines. arXiv:2101.12691 [cs.NI]

[121] Jagath Weerasinghe, Raphael Polig, Francois Abel, and Christoph Hagleitner. 2016. Network-attached FPGAs for data center applications. In *2016 International Conference on Field-Programmable Technology (FPT)*. 36–43. https://doi.org/10.1109/FPT.2016.7929186

[122] Xilinx. 2021. Alveo U25 SmartNIC Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u25.html (Accessed: Jun. 2021).

[123] Xilinx. 2021. UltraScale+ Devices Integrated 100G Ethernet Subsystem v3.1. https://www.xilinx.com/support/documentation/ip_documentation/cmac_usplus/v3_1/pg203-cmac-usplus.pdf (Accessed: Jun. 2021).

[124] Xilinx Inc. 2018. SDNet Development Environment. https://www.xilinx.com/products/design-tools/software-zone/sdnet.html (Accessed: Sep. 2018).

[125] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. 2014. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro* 34, 5 (Sept. 2014), 32–41. https://doi.org/10.1109/MM.2014.61