The I/O Driven Server: From SmartNICs to Data Movement Controllers

Justine Sherry
sherry@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

Many researchers are turning to SmartNIC offloads to improve the performance of high-performance networked systems. In this editorial, I discuss why SmartNICs are an especially powerful form factor for improving I/O intensive applications, and how their position in the dataplane enables them to take on central role in managing I/O. Rather than focusing on the benefits of individual offloads, this paper aims to explore the position of SmartNICs in the overall system integration of datacenter servers at the hardware and software level. I argue that SmartNICs should be viewed as 'data movement controllers' (NIC-DMCs) which are responsible for tasks involved in moving data between network, CPU, accelerators, and other endpoints: multiplexing/steering, interfacing between protocols, and enforcing I/O policies. I then enumerate open questions in how the hardware and software systems of the future will evolve to accommodate a dedicated NIC-DMC which is independent of the CPU complex.

1 Introduction

A large class of *I/O-Driven* applications are pushing the limits of what traditional server architectures can support in terms of throughput. These applications include network functions, key-value stores, caches, web servers, and microservices, and all of them combine a few key properties. First, they are driven by a high rate of incoming data: each piece of data arriving off of the network is implicitly a request for action by the server, and the arrival rate of this data is very high – up to 400Gbps on modern NICs. Second, they perform a relatively small amount of compute per byte of data. Unlike, say, scientific simulations (where a small amount of data can result in days of number crunching by one compute device), each network-triggered task is expected to complete in milliseconds or even microseconds (*i.e.* they have very low latency demands).

There is a growing sense that 'smart' NICs (or DPUs, or IPUs [49]), which inject some amount of programmable processing at the network interface card, are part of the solution to meet these challenging performance demands. To this end, we have seen a range of remarkable networked systems which use SmartNICs to 'offload' I/O intensive processing such as TCP and transport logic [6, 43], serialization [39], packet filtering [16, 51], network virtualization [12], and application multiplexing [16, 18, 24, 42]. Many of the papers published in this space show exciting throughput and latency gains going well past what was previously thought possible with the CPU-only approach.

It is tempting to read this literature and take away the lesson that this is yet another chapter in the lengthy story of how hardware acceleration can help improve application performance in a post-Moore era. Many SmartNICs offer *unconventional compute* offerings like PISA pipelines, FPGAs, or network processors (NPUs) and there is some evidence that for many workloads, these types of processors can outperform traditional processor cores in terms of throughput per Watt or latency. While the benefits of accelerators to improve efficiency are certainly *part* of the story of why SmartNICs are beneficial, to stop at this observation is a mistake.

In reality, there is something quite important about the NIC form-factor itself, not just the type of compute or acceleration that it offers. *The power of the SmartNIC is that it moves some compute to the 'front door' of the server, where an independent platform can now take over control of data movement of the server.* Taken to its logical conclusion, processor cores shift from being 'central' processing units – responsible for steering and orchestrating data movement within the server – and instead are liberated to focus only on application-specific operations.[1]

To understand what is happening, we can start by looking at two classes of processing which occur when data arrives from the network demanding to be processed. One class of activity is the true *data processing*: this is the application-specific activity, such as performing a key-value lookup in memory, executing a middlebox pipeline, or executing an RPC. Undergirding the true data processing, however, are *data movement operations* such as multiplexing incoming requests across hardware and processes, interfacing with protocols such as TCP and re-formatting data for the wire (serialization), or implementing policies and enforcing ACLs.

Historically, processor cores have been responsible for both data processing and managing data movement. Today, we are seeing a *physical* separation in the hardware to independently implement *data processors* and a *data movement controller.* SmartNICs take on the role of *data movement controllers* (DMCs), where processor cores as well as other hardware like accelerators take on the role of *data processors.* The goal of this document is to explore the new challenges in system integration across both software and hardware as SmartNICs (or DPUs) take over the role of data movement controller. Rather than considering individual tasks that we might 'accelerate', the goal is to envision the big picture in how dedicated data movement controllers will transform the architecture of future datacenters.

In §2 I will present three categories of tasks that fall under controlling data movement, and why I/O-intensive systems benefit from allowing a programmable NIC to take them over. These categories are multiplexing and steering data (§2.1), interfacing with the 'outside world' (§2.2), and enforcing I/O policies (2.3).

[1]Thanks to Derek Chiou for articulating this distinction!

In the remainder of the paper, I will discuss systems level challenges towards realizing an integrated NIC-DMC. First and foremost, there remain important software systems questions about *control*. Although moving DMC functionality into the NIC clearly delineates responsibility at a hardware level for who manages data movement operations, at a software level, the question of control is entirely muddled. Which software layer *should* manage the functionality on the NIC-DMC: should it be applications, the operating system, or a hypervisor? I discuss models of software control in §3.

In §4, I discuss other systems-level challenges for designing a NIC-DMC which cross-cut both hardware design and software systems engineering. Answering these questions will provide insights not only into the design of systems (given the availability of richly programmable NIC hardware) but also into the design of servers integrated with programmable NICs (given the needs of the systems that those in this community design and implement).

Before moving forward, it is worth noting that the observations outlined within seem to have been arrived at by many in the research community simultaneously, with both academia and industry pushing on many insights as to how to develop a NIC-DMC architecture. Indeed, in §3 we will see how the NIC-DMC vision is in some ways shared between application developers and multi-tenant datacenter operators, but in other ways is subtly different. The goal of this document is simply to articulate the changes that are happening, identify why they are needed, and forecast a bit how future servers will like and the the challenges we will encounter as we build them.[2]

2 Key DMC Operations

To understand the role of the SmartNIC in serving as a data movement controller (DMC), we can first explore the three tasks implemented by a DMC and the bottlenecks they introduce when implemented on same CPU(s) responsible for data processing. These tasks are:

- Multiplexing at both the software and hardware level (§2.1)
- Interfacing with the outside world (§2.2)
- Intermediation and policy enforcement (§2.3)

2.1 Multiplexing

A single network cable is a shared resource for every process and thread running on the server. Steering traffic to the correct data processor is hence the first and foremost responsibility for the DMC. This both means steering traffic to the right *processor* (at the hardware layer) as well as making data available to the right *process or thread* (at the software layer). Traditional hardware and software systems require the CPU to intermediate incoming data to steer it to its final destination. Unfortunately, this extra 'hop' between NIC and the true receiving data processor introduces unnecessary extra latency and eats up compute cycles. Moving this multiplexing into a NIC-DMC allows the NIC to send data to its final consumer in the first place.

Multiplexing at the Software Layer: Multiplexing between applications is traditionally performed by the kernel, with packets for different sockets arriving interleaved from the network and the kernel responsible for doling packets out to their respective

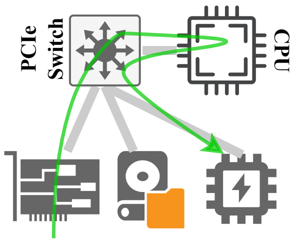

Figure 1: Although PCIe supports peer to peer operations, data from the NIC is typically routed through the CPU anyway.

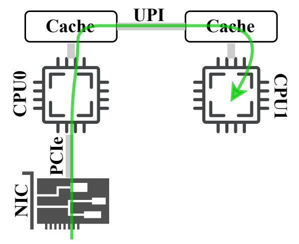

Figure 2: The NIC can only tansmit to its 'root' processor, and not directly to the processor core destined to process its data.

sockets. This approach has been widely lambasted for introducing expensive packet copies and context switches as data moves from kernel to application [36, 48].[3]

The adoption of kernel-bypass software designs effectively cut out this hop, pushing the issue of per-application multiplexing to the NIC. High-performance but otherwise 'dumb' NICs now support SR-IOV [35] and multiqueuing, RSS (which load balances packets across receive queues using a hash function), and/or FlowDirector [15] (which can use a basic match-action operation over the packet header to determine which receive queue to write incoming packets to). *By using these technologies to directly route packets to per-application receive queues, NICs took their first step towards taking responsibility of a data movement controller away from CPUs.*

Nonetheless, once packets are read into userspace from their respective receive queue, they may experience yet another round of multiplexing as packets are fanned out across worker threads. This might include another round of random hashing, a load-based algorithm such as join-shortest-queue, or an assignment based on stateful affinity (*e.g.*, mapping a request in a key-value store with a sharded keyspace). Consequently, applications might employ a dedicated 'I/O thread' to do this multiplexing – once again giving us a latency hit and wasting CPU cycles. Pioneering work such as FlexKVS [18] and RingLeader [24] (and many others) have shown that pushing this additional multiplexing task into the NIC provides worthwhile efficiency gains.

Multiplexing at the Hardware Layer: Unfortunately, even after software has patched its multiplexing inefficiencies by pushing multiplexing to the NIC-DMC, server hardware reinforces unnecessary multiplexing hops. These inefficiencies manifest in NUMA server architecture, and also whenever accelerators are the final data processors. The problem occurs because the root CPU hardware must intermediate data movement between the NIC and the destination CPU (which may be on a secondary NUMA node) or accelerator.

In Fig. 1, data arriving from the NIC is sent to to host memory and the CPU must explicitly send requests to an accelerator. This type of scenario might play out, for example, in an image classification microservice, in which an image arrives over the network for classification to be performed by a machine learning accelerator. In

[2]This manuscript is an extension of my keynote at EuroP4 2022 [44]. With apologies to Scott Shenker for stealing his catchphrase, 'There is nothing new here.'

[3]When there are multiple resident VMs, a second layer of DMC operations occurs in the hypervisor with similar overheads and an equivalent movement to avoid them by giving VMs direct access to network devices, which may be virtualized in hardware [35].

Fig. 2, data arriving from the NIC is copied into host memory attached to its PCIe 'root' CPU – but the core destined to consume the data is on another NUMA node. Hence, the data requires an additional 'hop' over QPI/UPI to reach the receiving processor; although no processor cycles are wasted by this procedure it nonetheless increases latency and leads to throughput loss. This overhead is is observable in any multicore application attempting to naïvely scale out across multiple NUMA nodes. [4]

We can see today's SmartNICs take on these *hardware multiplexing* challenges in several exciting designs in the literature today. NVIDIA's GPUDirect [33] allows a NIC to read/write directly to GPU memory, *e.g.* facilitating the classification example described above. In the research literature, IO-TCP [21] enables direct disk access by the NIC, accelerating services such as file transfer applications or video streaming which transfer files to clients over the Internet. And, addressing the NUMA challenge illustrated in Figure 2, IOctopus [45] connects a single NIC-DMC to multiple PCIe interfaces, allowing a single NIC to communicate point-to-point with multiple NUMA roots.

Lessons for a NIC-DMC: By taking on multiplexing at both the software and hardware layer, a NIC-DMC can send traffic from the network directly to its receiving data processor, without any intermediary steering or switching performed by the CPU. Multiplexing on the NIC-DMC can hence reduce latency (by avoiding extra hops) and leave more CPU cycles for application layer processing.

2.2 Interfacing

Another core task in any data movement procedure is *interfacing* between the sending and receiving platforms. Data is constantly packaged and re-packaged, marshalled and de-marshalled, from data structures into wire formats, into TCP packets and Ethernet frames, *etc.*. Traditionally, the CPU is required to spend cycle time performing this re-packaging as it intermediates transfers of data between applications, accelerators, disk, the network, *etc.* When the NIC takes on a role as a data movement controller, it must take on these tasks for *correctness* of multiplexing, but also may benefit from doing so for performance reasons.

Interfacing for Accelerators: When a DMC-NIC takes on the responsibility of steering traffic directly to/from accelerators, it also must take on the role of re-formatting data between wire formats for the network and the format expected by accelerators. Accelerators have their own host of data formats: they may consume or produce 32-bit or 64 bit words; they may consume streams of data, or fixed sized vectors; *etc.*

It is an unreasonable design choice to expect every accelerator to support the complex protocols that are used over external networks. Implementing, *e.g.*, TCP support, adds significant complexity to any hardware design and it would be inefficient to add such support to each and every device attached to a server when we might instead do so centrally at one location – namely, the DMC.

To the extent that sort of translation is supported today in practice, it is used with RDMA or RDMA-like protocols in which both the client and the server have been specially configured for exchange; technologies like NVIDIA's GPUDirect [33] take this approach. For Internet-facing or more general purpose applications, more research will be needed to design NIC-DMCs which are capable of interfacing between accelerators and the network. IOTCP – referenced above for enabling direct transfer of data from disk to network – is a very nice model in which a NIC-DMC intermediates communication between a disk (operating on a block abstraction) and the network (operating on an HTTP/TCP abstraction).

Interfacing for Applications: When the DMC performs multiplexing for accelerators, it must perform interfacing for the accelerators to do their work correctly. It is also the case that the DMC must implement some higher-layer protocols to do its own job correctly.

Consider a multicore key-value store in which data has been sharded with ranges of data assigned to each core in order to avoid contention and race conditions. When the DMC decides to which processor core's queue an incoming GET request belongs, it must inspect up the network stack into the application layer to identify the object ID being requested. If a request for an object comes over a TCP connection, this means that the DMC must parse the TCP packet, perform flow reassembly, and then parse the key-value store's Layer 7 messaging protocol, all to finally identify the object being requested and to steer the incoming request to the core with ownership of that object.

Hence, in order to correctly perform application-layer multiplexing, the DMC must be aware of TCP and the Layer 7 protocol for the key-value sture. Since this functionality is already implemented at the DMC, it makes little sense to re-do this work again at the receiving data processor.

Interfacing for Performance Gains: While the above two arguments focus on the need to implement interfacing functionality for *correctness* once the NIC has taken on multiplexing, there is also a performance argument for doing so! Many refer to interfacing tasks as part of a broader 'datacenter tax' [46] and argue that these operations should be offloaded from the CPU to improve performance and efficiency; such offloads include checksum verification, segmentation offloads [47], full TCP offloads [6, 43], TLS processing [34], serialization offloads [39], or network virtualization [12].

It is worth taking a pause to understand *why* it might be a good idea, from a performance perspective, to push this functionality into the NIC. Why should simply taking a task and moving it somewhere else improve performance?

One reason is that SmartNICs often offer unconventional hardware to implement these features, such as network processing units (NPUs) [30, 32], PISA pipelines [5, 25], FPGAs [14, 26], or network-specific accelerators [28]. Sometimes, but not always, these platforms can perform the same computation as a traditional von Neumann CPU with lower latency, lower jitter, or higher throughput per Watt of energy. Understanding when and how alternative computation platforms 'beat' traditional processors remains an active area of research.

Another reason – especially in multitenant datacenters – that moving this interfacing functionality to the NIC might be useful, for its own sake, is that it preserves the primary CPU cores for other uses, namely tenant compute. If the maximum core-count

[4]For this reason, many high-performance applications will exclusively use cores on the primary/root NUMA node – effectively leaving 1/2 of the potential capacity on the table.

for a CPU in a datacenter is 24 cores, the operator can sell at most 23-core VMs to their tenants, if they require a dedicated core to implement, *e.g.*, interfacing for virtual networking. By moving this compute to a separate platform entirely, the maximum marketable VM size is now 24 cores [12].

Lessons for a NIC-DMC: To correctly support multiplexing in a NIC-DMC, the NIC must be capable of taking on interfacing tasks such as Ethernet, IP, TCP, and even serialization support. In addition, there may often be performance and efficiency gains to pushing functionality into a NIC-DMC as unconventional hardware such as NPUs, PISA pipelines, and FPGAs can often implement such tasks at higher throughputs and lower latencies with less energy cost.

2.3 Policy Intermediation

A third category of functionality that the DMC is tasked with is *policy intermediation* such as firewalling, implementing QoS or rate limiting, or virtualizing network addressing. These tasks are implemented by a privileged system component which enforces that this functionality is applied over every application and service. Historically, these features were co-resident with the multiplexing and interfacing functionality embedded in the kernel or hypervisor.

Today, many major datacenters have offloaded many of their hypervisor policies to 'smart' hardware, *e.g.*, Microsoft's Accelnet [12], AWS Nitro [4], VMware's Project Monterey [10], and Alibaba's Fidas [9]. Hence, this *policy intermediation* category is (arguably) the most successful of the three classes of DMC functionality moving onto SmartNIC hardware.

Physical Rather than Virtual Isolation: Unlike many DMC interfacing tasks like TCP support which can be implemented in user-space, these operations are fundamentally privileged – one cannot, *e.g.* expect an unruly application developer to enforce rate limiting upon themselves. Guaranteeing that an administrator-set rate limiting policy is enforced correctly requires an administrator-privileged execution environment. Hence, implementing these features on a traditional processor requires context switching, which eats cycles and hurts latency. On the other hand, implementing these features on a physically separate device uses physical isolation to enforce the policy, without this overhead.

Intermediation for Accelerators: Most accelerators lack any sort of protected or privileged mode and hence an external DMC – CPU or NIC – must do this work for them. If we have moved mulitplexing to the NIC, then we must move these operations as well.

Performance Gains: As with interfacing tasks above, the same arguments about the benefits of unconventional hardware apply to offloading these tasks from CPU to NIC for performance reasons.

Lessons for a NIC-DMC: NIC-DMCs can provide physical, rather than virtual, isolation to enforce policies over network I/O, avoiding the cost of context switching on the data processor. NIC hardware may also be able to improve performance efficiency of these operations by performing policy functionality like firewalling, rate limiting, *etc.* in unconventional hardware.

3 Managing a NIC-DMC

In the previous section, we outlined three categories of data movement functionality that are moving out of 'Central Processing Units' and into dedicated 'Data Movement Controllers' embedded in SmartNICs: (1) multiplexing/steering, (2) interfacing, and (3) policy enforcement. Many academic prototypes focus on a particular task in just one of these categories – *e.g.*, implementing firewalling [17], or application-specifici steering [24]. However, the overall trajectory is towards an *integrated* DMC which performs all three classes of functionality, fully freeing processor cores from orchestrating data movement altogether.

However, how this integration will occur remains hazy. A key question for both software systems developers and hardware developers is *who will manage the DMC* (we will discuss other integration challenges in §4). The problem is that multiplexing, interfacing, and policy enforcement in the traditional model happen at three levels of privilege: that of the application, the operating system, or the hypervisor. Today, different subcommunities in academia *and* industry are making very different assumptions about 'who' controls the DMC.

There are good reasons that application developers, OS developers, and hypervisor developers all want a stake in what happens on the DMC. *Applications* perform multiplexing when they fan out, *e.g.* incoming HTTP requests across threads, and they perform interfacing when implementing L7-ish protocols like QUIC or TLS. *OS kernels* perform multiplexing when they steer TCP and UDP data to the appropriate application layer socket; they perform interfacing when they implement protocols like TCP, UDP, and Ethernet; they perform policy enforcement with tools like `netfilter` in Linux. And finally, *hypervisors* perform multiplexing when they steer incoming packets to the correct virtual machine, interfacing when they implement virtual networking, and they perform policy enforcement when they enforce rate limits or firewalling.

So who will be programming the NIC-DMC in the future?

There are a few competing visions today, each of which picks *either* the application, the OS, or the hypervisor as the logical owner of the NIC-DMC.

The Hypervisor as Owner: The most sophisticated artifacts towards an integrated NIC-DMC are being pushed by cloud operators through systems like Microsoft's Project Boost [7], VMware's Project Monterey [10], and Amazon's AWS Nitro [4]. These systems perform VM multiplexing on a NIC-DMC, interfacing for virtual networking, and policy enforcement such as firewalling and rate limiting. Increasingly they are also implementing interfacing for other forms of I/O, such as network attached storage [10]. Indeed, Microsoft, VMWare, and Amazon have all either declared success or an intent to move *all* hypervisor functionality off of processor cores and onto a NIC-DMC, enabling them to sell 'virtual' machines which are extremely close to the bare metal as the processor cores no longer need to be shared by the hypervisor and guest operating system.

The Application as Owner: At another extreme, many NIC-DMC deployments are entirely application-driven. These deployments tend to be used in environments where (a) servers are dedicated to a particular application, (b) this application demands high throughput and low latency I/O, and (c) there is a single administrator such that it is considered secure to bypass the operating system (*e.g.* using DPDK) and forgo using a hypervisor altogether. Pigasus is an academic Intrusion Detection Systems (IDS) in which the application assumes complete control of the NIC-DMC, programming it both with DMC functionality (Ethernet and TCP interfacing, multicore load-balanacing) as well as some IDS-specific functionality [51]. In industry, some of the most high-performance and tightly integrated application-controlled deployments are found in trading firms. For example, Jane Street uses SmartNICs to For example, Jane Street uses SmartNICs to implement filtering and flow steering of high-bandwidth marketdata feeds [1].

The Operating System as Owner: A third path – with significantly less deployment traction than the hypervisor-controlled or application-controlled models – is that the operating system might take control of the functionality on the NIC-DMC. An argument in favor of this approach comes from my student Hugo Sadok in his work on Kernel On-Path Interposition (KOPI) [40]. Hugo observes that the operating system traditionally does a tremendous amount of DMC work, from multiplexing incoming TCP data between sockets; implementing protocol support for Ethernet, TCP, IP, etc; to enforcing administrative policies via tools like `iptables` and `qdisc`. Today, many high performance applications have adopted kernel-bypass designs to achieve high throughput and low latency, but in the process, given up the benefits of all of this traditional kernel functionality. Hugo argues that the only way to get this functionality back, without sacrificing the performance benefits of kernel-bypass, is to *physically* maintain the kernel-bypass architecture (allowing the NIC to transfer data directly to userspace and vice versa) while *logically* restoring the kernel into the datapath by giving the kernel control over the functionality implemented on the NIC-DMC.

While the KOPI approach brings a missing piece of the traditional data movement story (that of the operating system) back into the picture for the future of SmartNICs, it also suffers the same drawbacks as the application-controlled and the hypervisor-controlled approaches. Specifically, each of these approaches preclude the others: they hypervisor and OS approaches do not have a 'story' for how to integrate application-layer offloads, and the application-layer approach completely bypasses any priveleged layer for enforcing policies from OS administrators or datacenter operators.

A hybrid approach?: A natural idea is to provide some sort of hybrid, multi-programmable approach: in which the NIC-DMC contains 'hooks' for programming by applications, operating systems, *and* hypervisors. Very nascent but exciting work exists to explore this space [8, 19]. The complexity of enabling multiple programmers on the NIC-DMC varies depending on the programmable hardware on the NIC. For example, NICs hosting traditional processor cores might require protection rings to isolate instructions from each programmer at different layers of privilege [41]. However, NICs using spatial compute platforms like FPGAs (or CGRAs) might support *tiling* in which the physical space of the device is allocated to different programmers [19].

Ultimately, the danger with a hybrid approach is that, in implementing the isolation and multiplexing mechanisms to support programming by applications, OSes, *and* hypervisors, we might re-introduce many of the undesirable performance overheads we seek to avoid in modern high-I/O systems. We may also dramatically increase resource requirements (*e.g.*, requiring more memory to host per-application policies for dozens of co-resident applications). Perhaps the cost is too high, and an approach with a single, privileged programmer sitting at the hypervisor will win out.

Why These Abstractions Matter: Ultimately, the question of *who* will be programming the NIC-DMC is a question for software systems thinkers, not hardware research. The needs we have at the software layer should shape the future of the hardware designs that NIC vendors offer. For example, if we want multiprogramming support for the NIC-DMC, the hardware offerings may have to change in order to enable isolation or process switching. NICs designed with hypervisor use cases in mind may not support deep packet inspection (since hypervisors rarely interact with packet contents), where NICs designed with application developers needs in mind might come with 'hardened' support for TCP reassembly and a flexible DPI engine.

Today, hyperscalers like Microsoft, Google, and Amazon are the largest consumers of SmartNICs/DPUs, and their hypervisor-centric use cases drive the discussion about the future hardware architecture of these devices. If the rest of the software system community fails to articulate what functionality we want or need, we may arrive at a future where available NIC-DMC hardware is not able to support the use cases that our explosively creative research and development community envisions today.

4 Future Challenges

The research community has identified a plethora of exciting use cases for SmartNICs that fall under the categories of multiplexing, interfacing, and policy management. Significant research remains in how to implement those things well using the unconventional hardware offered by these devices. However, the purpose of this article has been to 'up-level' the conversation around SmartNICs to focus on the *shift in control* as SmartNICs move to take on the role of 'data movement controller' and relieve processor cores of multiplexing, interfacing, and policy management altogether.

In the previous section, we spoke of one question that I think is critical towards envisioning the future of the NIC-DMC: the question of which priveleged entities will be allowed to program and manage the DMC. However, there are many other important challenges that arise as we start to think of the NIC-DMC as an integrated controller rather than a simple offload platform. Hence, before concluding I sketch some additional questions about the future integration of DMCs into the hardware and software of I/O intensive servers.

What is the right ensemble of compute on the NIC-DMC? Today's SmartNICs offer a range of programmable hardware offerings (*e.g.* FPGAs [37], PISA pipelines [5], network processor [30, 32], or x86 cores [11]). They also offer task-specific accelerators, *e.g.* for cryptography support or compression [31]. While at first glance, this seems like a hardware question, it is also a software question as *what is needed depends on what the software demands of the platform.*

Networked systems researchers should not shy away from making statements about what is needed based on their experiences building DPU-driven applications. PANIC [25], for example, provides a nice example of networking researchers providing the insight that multiple compute platforms will be needed on future SmartNICs and that switching should hence be a core component of future NIC offerings.

How should the server interconnect change to support a NIC-DMC? PCIe implementations are fundamentally orchestrated towards the idea that processor cores are a 'central processing unit' coordinating data movement. Most network topologies have a 'root' with one processor designated as the root. The PCIe protocol historically only had support for dedicated 'peripherals' to communicate with the 'root', and although modern versions of the protocol support 'peer to peer' communication (*e.g.* between NIC and disk) many devices simply do not support this feature [2]. Finally, PCIe today is prone to congestion and known to have high latencies. While there are alternative technologies such as CXL [3] (which is based in PCIe anyway) and NVLink, they too inherit some of these problems. There is an open field for the design of a *new* server interconnect, one which recognizes the shift of DMC operations away from processor cores and on to a dedicated device. Once again, this may at first glance seem like a hardware/architecture question, but surely networking researchers have insights to offer in the design of what is, afterall, another network.

What does the shift towards a DMC mean for security? Every time systems researchers propose an exciting new thing, security researchers come in, spoil our party, and tell us we have created a massive vulnerability somehow.

Where are the 'open' platforms and code to support integration? Systems researchers benefit tremendously from building and extending each other's platforms, with a legacy from BSD [38] to AFS [29] to Spark [50]. In networking, the Click software router [22] catalyzed a huge advance in developing novel applications and research prototypes. A few proposals exist in the literature – especially in the FPGA space where there are open-source systems like Rosebud [20], ClickNP [23], Corundum [13], and Fluid [27] – but we have yet to see massive traction and a community pushing new features into a shared system.

5 Conclusion

The power of compute at the network interface card to improve performance and efficiency is now a well-established success story. Our next step as a community is to stop thinking of SmartNICs as a platform for 'offloads', and instead to think of them as a platform with a clearly defined *role* in the software and hardware architecture. In this paper, I defined this role as a 'data movement controller', with a clearly defined scope of responsibilities: multiplexing, interfacing, and policy enforcement. Nonetheless, the shape of the role of future DMCs remains open for discussion, and we as a community should continue to articulate what we do and do not require from this platform.

In defining the *role* of the NIC-DMC, we also set the stage for operating systems, applications, and hypervisors to re-organize themselves around the existence of this platform. Importantly, we in the software systems community *must* articulate what functionality we need from the DMC – as this in turn should drive the future hardware architecture of these nascent devices.

Acknowledgements

Thanks to Hugo Sadok, Nirav Atre, Aurojit Panda, Boon Ang, Brian Nigito, Ron Minsky, Ren Wang, Derek Chiou, and James Hoe for the interesting discussions and useful feedback on this draft. This work was supported by the Intel/VMware Crossroads 3D FPGA Research Center, a VMware Systems Research Award, and a Google Research Gift.

References

[1] 2023. *Private communciation with Brian Nigito and Ron Minsky.*

[2] 2023. *Private communciation with Ren Wang.*

[3] 2023. Compute Express Link. https://www.computeexpresslink.org/

[4] Amazon. 2023. AWS Nitro System. https://aws.amazon.com/ec2/nitro/

[5] AMD Corporation. [n. d.]. AMD Pensando SmartNIC. https://www.amd.com/en/accelerators/pensando

[6] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling Programmable Transport Protocols in High-Speed NICs. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'20)*. USENIX Association, USA, 93–110.

[7] Microsoft Azure. 2023. Preview: Azure Boost. https://azure.microsoft.com/en-us/updates/preview-azure-boost/

[8] 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

[9] Jian Chen, Xiaoyu Zhang, Tao Wang, Ying Zhang, Tao Chen, Jiajun Chen, Mingxu Xie, and Qiang Liu. 2022. Fidas: Fortifying the Cloud via Comprehensive FPGA-Based Offloading for Intrusion Detection: Industrial Product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 1029–1041. https://doi.org/10.1145/3470496.3533043

[10] Kit Colbert. 2020. Announcing Project Monterey: Redefining Hybrid Cloud Architecture. VMware Blog.

[11] Kevin Deierling. 2020. What Is a DPU? NVIDIA Blog. https://blogs.nvidia.com/blog/2020/05/20/whats-a-dpu-data-processing-unit/

[12] 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 *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation* (Renton, WA, USA) *(NSDI'18)*. USENIX Association, USA, 51–64.

[13] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps Nic. In *28th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2020, Fayetteville, AR, USA, May 3-6, 2020*. IEEE, 38–46. https://doi.org/10.1109/FCCM48280.2020.00015

[14] Intel Corporation. [n. d.]. Infrastructure Processing Unit. https://www.intel.com/content/www/us/en/products/details/network-io/ipu.htm

[15] Intel Corporation. 2023. Introduction to Intel Ethernet Flow Director and Memcached Performance. Whitepaper 331109-001US.

[16] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, JR. Gerald Q. Maguire, and Rebecca Steinert. 2021. Metron: High-Performance NFV Service Chaining Even in the Presence of Blackboxes. *ACM Trans. Comput. Syst.* 38, 1–2, Article 3 (jul 2021), 45 pages. https://doi.org/10.1145/3465628

[17] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, JR. Gerald Q. Maguire, and Rebecca Steinert. 2021. Metron: High-Performance NFV Service Chaining Even in the Presence of Blackboxes. *ACM Trans. Comput. Syst.* 38, 1–2, Article 3 (jul 2021), 45 pages. https://doi.org/10.1145/3465628

[18] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. *SIGARCH Comput. Archit. News* 44, 2 (mar 2016), 67–81. https://doi.org/10.1145/2980024.2872367

[19] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J. Rossbach. 2018. Sharing, Protection, and Compatibility for Reconfigurable Fabric with AmorphOS. In *13th USENIX Symposium on Operating*

Systems Design and Implementation (OSDI 18). USENIX Association, Carlsbad, CA, 107–127. http://www.usenix.org/conference/osdi18/presentation/khawaja

[20] Moein Khazraee, Alex Forencich, George C. Papen, Alex C. Snoeren, and Aaron Schulman. 2023. Rosebud: Making FPGA-Accelerated Middlebox Development More Pleasant. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 586–605. https://doi.org/10.1145/3582016.3582067

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

[22] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. 2000. The Click Modular Router. *ACM Trans. Comput. Syst.* 18, 3 (aug 2000), 263–297. https://doi.org/10.1145/354871.354874

[23] 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)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/2934872.2934897

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

[25] 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

[26] Mangoboost. 2023. Mangoboost DPU Accelerator. https://mangoboost.io/

[27] Joseph Melber. 2022. Fluid: Raising the Level of Abstraction for FPGA Accelerator Development Without Compromising Performance. (6 2022). https://doi.org/10.1184/R1/19787158.v1

[28] Mellanox Corporation. [n. d.]. Mellanox ConnectX-5. https://www.nvidia.com/en-us/networking/ethernet/connectx-5/

[29] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. Rosenthal, and F. Donelson Smith. 1986. Andrew: A Distributed Personal Computing Environment. *Commun. ACM* 29, 3 (mar 1986), 184–201. https://doi.org/10.1145/5666.5671

[30] Netronome. [n. d.]. Netronome Agilio SmartNICs. https://www.netronome.com/

[31] NVIDIA. 2023. ConnectX SmartNICs. https://www.nvidia.com/en-us/networking/ethernet-adapters/.

[32] NVIDIA Corporation. [n. d.]. Bluefield Data Processing Unit. https://www.nvidia.com/en-us/networking/products/data-processing-unit/

[33] NVIDIA Corporation. 2023. NVIDIA GPUDirect. https://developer.nvidia.com/gpudirect

[34] NVIDIA Corporation. 2023. TLS Offload using NVIDIA Bluefield DPU. https://docs.nvidia.com/doca/sdk/tls-offload/index.html

[35] Peripheral Component Interconnect Special Interest Group. 2015. Root Complex Integrated Endpoints and IOV Updates. PCI-SIG Document 11110.

[36] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2014. Arrakis: The Operating System is the Control Plane. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 1–16. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/peter

[37] 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. 2015. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *IEEE Micro* 35, 3, 10–22. https://doi.org/10.1109/MM.2015.42

[38] John S. Quarterman, Abraham Silberschatz, and James L. Peterson. 1985. 4.2BSD and 4.3BSD as Examples of the UNIX System. *ACM Comput. Surv.* 17, 4 (dec 1985), 379–418. https://doi.org/10.1145/6041.6043

[39] Deepti Raghavan, Shreya Ravi, Gina Yuan, Pratiksha Thaker, Sanjari Srivastava, Micah Murray, Pedro Penna Henrique, Amy Ousterhout, Philip Levis, Matei Zaharia, and Irene Zheng. 2023. Cornflakes: Zero-Copy Serialization for Microsecond-Scale Networking. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*.

[40] Hugo Sadok, Zhipeng Zhao, Valerie Choung, Nirav Atre, Daniel S. Berger, James C. Hoe, Aurojit Panda, and Justine Sherry. 2021. We Need Kernel Interposition over the Network Dataplane. (May 2021).

[41] Michael D. Schroeder and Jerome H. Saltzer. 1972. A Hardware Architecture for Implementing Protection Rings. *Commun. ACM* 15, 3 (mar 1972), 157–170. https://doi.org/10.1145/361268.361275

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

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

[44] Justine Sherry. [n. d.]. Re-envisioning Generic Server Architectures for I/O-Driven Compute. Keynote at EuroP4 2022 Workshop. https://www.youtube.com/watch?v=Lo0mVet4eZM

[45] Igor Smolyar, Alex Markuze, Boris Pismenny, Haggai Eran, Gerd Zellweger, Austin Bolen, Liran Liss, Adam Morrison, and Dan Tsafrir. 2020. IOctopus: Outsmarting Nonuniform DMA *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 101–115. https://doi.org/10.1145/3373376.3378509

[46] Akshitha Sriraman and Abhishek Dhanotia. 2020. Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale. 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, 733–750. https://doi.org/10.1145/3373376.3378450

[47] The Linux Kernel documentation. [n. d.]. Segmentation Offloads. https://docs.kernel.org/networking/segmentation-offloads.html

[48] T. von Eicken, A. Basu, V. Buch, and W. Vogels. 1995. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. *SIGOPS Oper. Syst. Rev.* 29, 5 (dec 1995), 40–53. https://doi.org/10.1145/224057.224061

[49] Wikipedia. 2023. Data processing unit — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Data%20processing%20unit&oldid=1164863994. [Online; accessed 04-August-2023].

[50] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (San Jose, CA) *(NSDI'12)*. USENIX Association, USA, 2.

[51] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James Hoe, Vyas Sekar, and Justine Sherry. 2020. Achieving 100Gbps Intrusion Prevention on a Single Server. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI '20)*. USENIX Association, Berkeley, CA, USA.