FlexNIC: Rethinking Network DMA

Antoine Kaufmann Simon Peter Thomas Anderson Arvind Krishnamurthy
University of Washington

Abstract

We propose FlexNIC, a flexible network DMA interface that can be used by operating systems and applications alike to reduce packet processing overheads. The recent surge of network I/O performance has put enormous pressure on memory and software I/O processing subsystems. Yet even at high speeds, flexibility in packet handling is still important for security, performance isolation, and virtualization.

Thus, our proposal moves some of the packet processing traditionally done in software to the NIC DMA controller, where it can be done flexibly and at high speed. We show how FlexNIC can benefit widely used data center server applications, such as key-value stores.

1 Introduction

Network bandwidth is growing steadily: 40 Gbps Ethernet is a commodity and 100 Gbps is starting to become available. This trend is straining the server computation and memory capabilities that we have to devote to process network traffic, and it is likely to limit future server performance. For example, last level cache access latency in Intel Sandy Bridge processors is 15 ns [24] and has not improved in newer processors. A 100 Gbps interface can receive a cache-line sized (64B) packet close to every 5 ns. Thus, at 100 Gbps, even a single last-level cache access in the packet data handling path will prevent software from keeping up with the tremendous speed of arriving network traffic.

We claim that the primary reason for high memory and processing overheads is the inefficient use of these resources by current commodity network interface cards (NICs). NICs communicate with software by accessing data in server memory, either in DRAM or via a designated last level cache (e.g., via DDIO [15], DCA [14], or TPH [28]). A number of packet descriptor queues instruct the NIC as to where in server memory it should place the next received packet and from where to read the next packet for transmission. Except for simple header splitting, no further modifications to packets are made and only basic distinctions are made among packet types. For example, it is possible to choose a virtual queue based on the TCP connection, but not, say, on the application key in a memcache lookup.

This design introduces overhead in several ways. For example, even if software is only interested in a subset of each packet, the current interface requires NICs to transfer packets in their entirety to host memory. No interface exists to steer packets to the right location in the memory hierarchy based on application-level information, causing extraneous cache coherence traffic when a packet is not in the right cache. Finally, network processing code repetitively has to process each packet of a network stream, such as to check packet headers, even if it is clear what to do in the common case.

The current approach to network I/O acceleration is to put fixed-function offload features into NICs [35]. While useful, these offloads bypass application and OS concerns and can thus only perform low-level functions, such as checksum processing and packet segmentation for commonly used, standard protocols (UDP and TCP). Higher-level offloads can constrain the system in other ways. For example, remote direct memory access (RDMA) [34], is difficult to fit to server applications [22, 9], resulting in diminished performance benefits [18]. Hardware I/O virtualization can eliminate hypervisor and operating system overheads [29, 4], but prevents common server consolidation techniques, such as memory overcommit, as DMA-able memory regions have to be pinned to ensure accessibility by the NIC.

To address these shortcomings, we propose FlexNIC, a flexible DMA interface for network I/O. FlexNIC allows applications and OSes to install packet processing rules into the NIC, which instruct it to execute simple operations on packets while exchanging them with host memory. These rules allow applications and the OS to exert fine-grained control over how data is exchanged with the host, including where to put each portion of it. FlexNIC can improve packet processing performance, while reducing memory system pressure at fast network speeds.

The idea is not far-fetched. OpenFlow switches support rule-based packet processing at high line-rates and are currently being enhanced with programmability features to allow transformations on packet fields [6]. High-speed programmable NICs that can support limited processing on the NIC also exist [25, 38, 8] and higher-level abstractions to program them are being proposed [16]. We build upon these ideas to provide a flexible, high-speed I/O interface for server systems.

The rest of this paper discusses what is needed to realize this vision—in hardware, the operating system, and applications—and points out several further use cases of the approach. For example, we show how FlexNIC can benefit key-value store performance by steering traffic based on the key in client requests to the cores with the corresponding value in their cache.

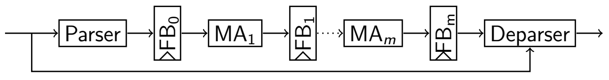

Figure 1: RMT switch pipeline.

2 Hardware Considerations

To provide the needed flexibility at fast line rates, we apply the reconfigurable match table (RMT) model recently proposed for flexible switching chips [7] to the NIC DMA interface. RMT offers packet processing performance an order of magnitude above specialized network processors [8, 25], at a cost well below an FPGA-based design [7]. The fact that RMT has been evaluated as feasible for switches in the current generation gives us confidence that the design can provide sufficient performance for the next generation of NICs. We briefly describe the RMT model in switches and then discuss how to apply it to arrive at an ideal NIC hardware model.

RMT in switches. RMT switches can be programmed with a set of rules that match on various parts of the packet, and then apply data-driven modifications to it as well as programmable routing, all operating at line rate for the switched packets. This is implemented using two packet processing pipelines that are connected by a set of queues allowing for packets to be replicated and then modified separately. Such a pipeline is shown in Figure 1. A packet enters the pipeline through the parser, which identifies all relevant packet fields as described by the software-defined parse graph. It extracts the specified fields into a field buffer (FB$_0$) to be used in later processing stages. The relevant fields pass through the pipeline of match and action (M+A) stages (MA$_1$..MA$_m$) and further field buffers (FB$_1$..FB$_m$). In a typical design, $m = 32$. An M+A stage matches on field buffer contents using a match table (of implementation-defined size), looking up a corresponding action, which is then applied as we forward to the next field buffer. Independent actions can be executed in parallel within one M+A stage. Finally, the deparser combines the modified fields with the original packet data received from the parser to get the final packet. To be able to operate at high line rates, multiple parser instances can be used. In addition to exact matches, RMT tables can also be configured to perform prefix matches or wildcard matches.

Applying RMT to NICs. To gain the largest benefit from our approach, we enhance commodity NIC DMA capabilities by integrating three RMT pipelines with the DMA engine, as shown in Figure 2. Similar to the switch model, incoming packets from any of the NIC's Ethernet ports (P$_1$..P$_n$) first traverse the ingress pipeline where they may be modified according to M+A rules. Packet fields can be added, stripped, or modified, potentially leaving a much smaller packet to be transfered to the host. Modified packets are stored in a NIC-internal packet buffer and pointers to the corresponding buffer positions are added to a number of NIC-internal holding queues depending on the final destination—host memory or P$_1$..P$_n$. From each holding queue, packets are dequeued and (depending on the final destination) processed separately in the *DMA pipeline* or the *egress pipeline*. Both can again apply modifications.

If host memory is the final destination, the DMA pipeline issues requests to the DMA controller for transferring data between host memory and packet, after which the packet can again be enqueued in NIC queues for further processing. DMA parameters such as the range in memory and the offset in the packet to transfer are passed to the DMA engine by adding a header to packets in the DMA pipeline. We can use this design to simply exchange packets with host memory or to carry out more complex memory exchanges. For example, to support RDMA-like protocols, a received RDMA read request can be passed to the DMA pipeline to get the data from the host and then sent out by enqueueing it to an egress pipeline to craft the response. Packet checksums can then be calculated on the final packets using the existing checksum offload capabilities.

Extensions. The design discussed so far provides the basis for efficiently interacting with host memory. Two simple extensions can be useful: 1. To implement stateful packet matching, we can allow M+A stages to match and modify small portions of NIC-internal SRAM. This extension has the potential to extend the functionality offered by existing NIC offloads, such as TCP segmentation offload, to other, application-defined protocols. 2. To enable protected application-level access to packet filtering under hardware virtualization [29, 4], we can limit virtual NIC (VNIC) access to only a subset of the RMT table space and assign an ID to this subset. For each VNIC, the OS is then free to insert ingress rules to tag a packet flow with the VNIC ID. FlexNIC should then ensure that VNIC rules have to first match their VNIC ID to restrict VNIC access to OS-defined network flows and to prevent conflicting rules from being inserted. VNICs can be safely exposed to applications by providing separate PCI virtual functions as enabled by SR-IOV [20].

Software interface. As a software interface to FlexNIC, we intend to use a modified version of the P4 language [6] proposed for configuring programmable switches that matches closely with the RMT model. Our modifications to P4 are minimal. An extension is required for handling modifiable memory in M+A stages, which requires additional actions for accessing this memory. Integration of the DMA controller, the other major change from RMT switches, does not require

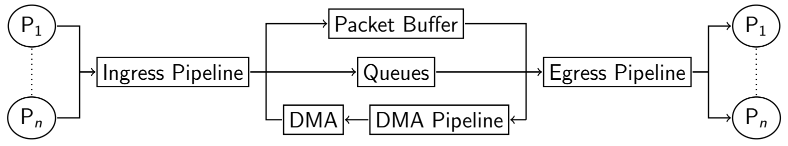

Figure 2: RMT-enhanced NIC DMA architecture.

Field	Description
`offset`	Byte offset in the packet
`length`	Number of bytes to transfer
`direction`	From/To memory
`memBase`	Start address in memory
`cache`	Cache level to send data to
`core`	Core Id, if data should go to cache
`atomicOp`	PCIe atomic operation [27]

Table 1: Header format for DMA requests.

language extensions, but we intend to add action verbs to simplify programming (cf. Figure 3). Communication with the DMA controller is implemented by adding a header to the packet, as shown in Table 1. P4 already includes the necessary support for doing so.

3 Application Integration

To demonstrate the benefits of FlexNIC, we show its integration with the popular Memcached key-value store [1]. Memcached is typically used to accelerate common web requests. In these scenarios, service latency and throughput are of utmost importance.

To scale request throughput, NICs offer receive-side scaling (RSS), an offload feature that directs incoming packets to descriptor queues based on client connection information. These queues are then dedicated to individual CPU cores to eliminate concurrent data access and scale performance with the number of clients. However, this approach suffers from a number of performance drawbacks [21]: 1) Hot items are likely accessed not just by one, but by all clients of the store, resulting in reduced cache utilization by replicating these items into all CPU caches. 2) When a hot item is frequently modified, global cache invalidations cause large cache traffic overheads.

FlexNIC allows us to tailor RSS to Memcached. Instead of assigning clients to server cores, we can partition the *key space* and use a separate *key space request queue* per core. We can install rules that steer client requests to appropriate queues, based on a hash of the requested key in the packet. The hash can be provided by the client or computed in the NIC. This approach maximizes cache utilization and minimizes cache coherence traffic. If a single core is not enough to serve a frequently accessed item, we can use FlexNIC to balance the load over available cores in a way we define. For example, two cores sharing a cache can benefit from low latency sharing.

Even if most client requests arrive well-formed at the server and are of a common type—say, GET requests—network stacks and Memcached have to inspect each packet to determine where the client payload starts, to parse the client command, and to extract the request ID. This incurs extra memory and processing overhead as the NIC has to transfer the headers to the host just so that software can check and then discard them. Measurements using the Arrakis OS showed that, assuming kernel bypass, network stack processing takes about a quarter of the total server processing time. Another quarter is spent in request/response processing in Memcached. The rest is to compute the hash function.

Using FlexNIC, we can check and discard Memcached headers directly on the NIC before any transfer takes place and eliminate up to half of the required server processing latency. To do so, we install a rule that identifies GET requests and transfers only the client ID and requested key to a dedicated fast-path request queue for GET requests. If the packet is not well-formed the NIC can detect this and instead transfer it in the traditional way to a slow-path queue for software processing.

To support all client hardware architectures Memcached has to convert certain packet fields from network to host byte order. We can instruct FlexNIC to carry out these simple transformations for us, while transferring the packets into host memory.

Finally, we can reduce memory system utilization due to NIC descriptor fetch and write-back by tailoring the data path interface to Memcached. For example, we can program a circular queue of fixed-size client read requests by allocating a range of memory for the queue inside Memcached with head and tail pointers. A rule can instruct the DMA controller to use a fixed stride over the memory range, computing the difference between the head and tail pointers to determine how much space is left in the queue. Head and tail pointers would only need to be accessed when we are low on queue space, instead of for every packet.

Figure 3 shows the resulting key-value store design, including a set of simplified FlexNIC rules that encapsulate these enhancements to steer GET requests for a range of keys to a specific core. The example excludes queue flow control for brevity.

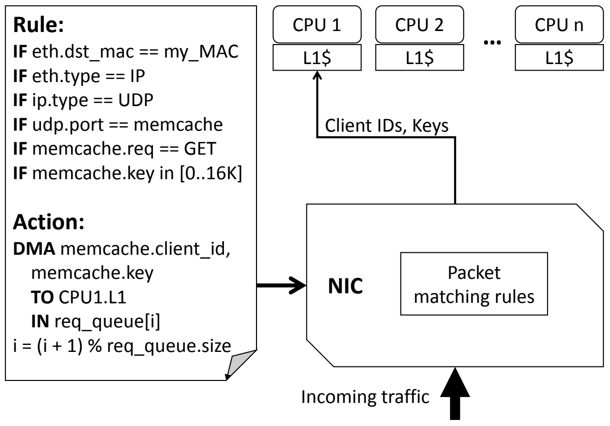

Figure 3: FlexNIC receive fast-path for Memcached: A rule matches GET requests for a particular key range, and then writes only the key together with a client identifier to a circular queue in memory.

4 OS Integration

So far we discussed our approach in the context of a single application with exclusive access to the NIC. In this section, we present two examples demonstrating that our approach can be efficiently integrated with the OS and that further benefits can arise when doing so.

Resource virtualization. Cloud services can be consolidated on a single server using resource virtualization. This allows the sharing of common hardware resources using techniques such as memory overcommit to save cost. However, traditional DMA operations require the corresponding host memory to be present, which is achieved by pinning the memory in the OS. With multi-queue NICs, kernel-bypass [29, 4], and RDMA, the amount of memory pinned permanently can be very large (cf. FaRM [9]), preventing effective server consolidation. Even without pinning, accesses to non-present pages are generally rare, so all that's required is a way to gracefully handle faults when they occur. Using FlexNIC, the OS can insert a rule that matches on DMA accesses to non-present regions and redirects them to a slow-path that implements these accesses in software. In this way these faults are handled in a manner fully transparent for applications.

Fast failover. User-level services can fail. In that case a hot standby or replica can be an excellent fail-over point if we can fail-over quickly. Using FlexNIC, an OS detecting a failed user-level application can insert a rule to redirect traffic to a replica, even if it resides on another server. Redirection can be implemented by a simple rule that matches the application's packets and then forwards incoming packets by rewriting the headers and enqueuing them to be sent out through the egress pipeline. No application-level modifications are necessary for this approach and redirection can occur with minimal overhead.

5 Further Use Cases

Beyond improving the performance of request-response applications, several further use cases emerge when using FlexNIC.

Consistent RDMA. RDMA provides low latency access to remote memory. However, it does not support consistency of concurrently accessed data structures. This vastly complicates the design of applications sharing memory via RDMA (e.g., self-verifying data structures in Pilaf [22]). FlexNIC can be used to enhance RDMA with simple application-level consistency properties. For example, when concurrently writing to a hashtable, FlexNIC could check whether the target memory already contains an entry by atomically [27] testing and setting a designated memory location and, if so, either defer management of the operation to the CPU or fail it.

Resource isolation. The ability to isolate and prioritize network flows is important in a range of load conditions [3]. A server receiving more requests than it can handle could decide to only accept specific request types, for example based on a priority specified in the request header. On servers that are not fully saturated, careful scheduling can reduce the average latency, for example by prioritizing cheap requests over expensive requests in a certain ratio and thereby reducing the overall tail latency. Achieving isolation implies minimizing the required processing resources, CPU cycles as well as memory, that need to be invested before a priority can be assigned or a request can be dropped [10, 23]. As network bandwidth increases, the cost of classifying requests in software can quickly become prohibitive. FlexNIC can be used to prioritize requests in hardware, and enqueue requests in different software queues based on priorities, or even reject requests, possibly after notifying the client.

Cache flow control. A common problem when streaming data to caches (found in DDIO-based systems) is that the limited amount of available host cache memory can easily overflow with incoming network packets if software is not quick enough to process them. In this case, the cache spills older packets to DRAM causing cache thrashing and performance collapse when software pulls them back in the cache to process them. To prevent this, we can implement a simple credit-based flow control scheme: The NIC can increment an internal counter for each packet written to a host cache. If the counter is above a threshold, the NIC instead writes to DRAM. Software acknowledges finished packets by sending an application-defined packet via the NIC that decrements the counter. This ensures that packets stay in cache for as long as software accesses them to keep performance stable.

Virtual switch. Data center network functions are increasingly migrating to the edge [31]. Formerly specialized services, such as load balancing, firewalling, and network analytics, are carried out on commodity servers using virtual software switches [2, 30]. This ensures performance scalability for the core of the network, while allowing innovation at software timescales at the edge. However, network processing is taking away an increasing number of server cores and memory bandwidth from virtual machines. While this is tenable at 10G speeds, it can quickly become prohibitive at 40G to 100G. FlexNIC can allow virtual switching to be offloaded to the NIC, while retaining the flexibility of software development cycles.

GPU networking. General purpose computation capabilities on modern GPUs are increasingly being exploited to accelerate network applications [13, 17, 36, 19]. So far all these approaches require CPU involvement on the critical path for GPU-network communication. GPUnet [19] exploits P2P DMA to write packet payloads directly into GPU memory from the NIC, but the CPU is still required to forward packet notifications to/from the GPU using an in-memory ring buffer. Further, GPUnet relies on RDMA for offloading protocol processing to minimize inefficient sequential processing on the GPU. With FlexNIC, notifications about received packets can be written directly to the ring buffer because arbitrary memory writes can be crafted, thereby removing the CPU from the critical path for receiving packets. In addition FlexNIC's offload capabilities can enable the use of conventional protocols such as UDP, by offloading header verification to hardware.

6 Related Work

NIC-Software co-design. Previous attempts to improve NIC processing performance used new software interfaces to reduce the number of required PCIe transitions [12] and to enable kernel-bypass [32, 37, 11]. RDMA goes a step further to provide remote direct memory access, entirely bypassing the remote CPU. Scale-out NUMA [26] extends the RDMA approach by integrating a remote memory access controller with the processor cache hierarchy that automatically translates certain memory accesses into remote memory operations. sNICh [33] accelerates switching of packets among virtual machines and the network by caching flow forwarding assignments in the NIC. Our approach builds upon these ideas to provide a more flexible application-level networking interface.

Programmable network hardware. In the wake of the software-defined networking trend, a rich set of customizable switch data planes have been proposed. For example, the P4 programming language proposal [6] allows users rich switching control based on arbitrary packet fields, independent of underlying switch hardware. We adapt this idea to provide similar functionality for the NIC-host interface.

Direct cache access. Data Direct I/O (DDIO) [15] attaches PCIe directly to an L3 cache. DDIO is software-transparent and as such does not easily support the integration with higher levels of the cache hierarchy. Prior to DDIO, direct cache access (DCA) [14] supported cache prefetching via PCIe hints sent by devices and is now a PCI standard [28]. With FlexNIC support, DCA tags could be re-used by systems to fetch packets from L3 to L1 caches. While not yet supported by commodity servers, tighter cache integration has been shown to be beneficial in systems that integrate NICs with CPUs [5].

7 Conclusion

As multicore servers scale, the boundaries between NIC and switch are blurring: NICs need to route packets to appropriate cores, but also transform and filter them to reduce software processing and memory subsystem overheads. FlexNIC provides this functionality by extending the NIC DMA interface.

FlexNIC also integrates nicely with current software-defined networking trends. For example, by offloading fine-grained server control, such as load balancing to individual server cores, to a programmable core network switch when this is more efficient: Core switches can tag packets with a decision that can be executed by FlexNIC.

We intend to demonstrate FlexNIC's efficacy by implementing a full prototype using a programmable NIC, such as the NetFPGA SUME [38].

Acknowledgements

This work was supported by NetApp, Google, and the National Science Foundation (CNS-0963754, CNS-1217597). We would also like to thank Taesoo Kim, Luis Ceze, Jacob Nelson, Timothy Roscoe, and John Ousterhout for their input on early drafts of this paper. Jialin Li and Naveen Kr. Sharma collaborated on a class project providing the basis for FlexNIC.

References

[1] `http://memcached.org/`.

[2] `http://www.vmware.com/products/nsx`.

[3] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *3rd USENIX Symposium on Operating Systems Design and Implementation*, 1999.

[4] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[5] Nathan L. Binkert, Ali G. Saidi, and Steven K. Reinhardt. Integrated network interfaces for high-bandwidth TCP/IP. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Computer Communication Review*, 44(3), 2014.

[7] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM Conference on SIGCOMM*, 2013.

[8] Cavium Corporation. *OCTEON II CN68XX Multi-Core MIPS64 Processors*. `http://www.cavium.com/pdfFiles/CN68XX_PB_Rev1.pdf`.

[9] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation*, 2014.

[10] Peter Druschel and Gaurav Banga. Lazy receiver processing (LRP): A network subsystem architecture for server systems. In *2nd USENIX Symposium on Operating Systems Design and Implementation*, 1996.

[11] Peter Druschel, Larry Peterson, and Bruce Davie. Experiences with a high-speed network adaptor: A software perspective. In *ACM Conference on SIGCOMM*, 1994.

[12] Mario Flajslik and Mendel Rosenblum. Network interface design for low latency request-response protocols. In *USENIX Annual Technical Conference*, 2013.

[13] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: A GPU-accelerated software router. In *ACM Conference on SIGCOMM*, 2010.

[14] Ram Huggahalli, Ravi Iyer, and Scott Tetrick. Direct cache access for high bandwidth network I/O. In *32nd Annual International Symposium on Computer Architecture*, 2005.

[15] Intel Corporation. Intel data direct I/O technology (Intel DDIO): A primer. `http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/data-direct-i-o-technology-brief.pdf`, February 2012. Revision 1.0.

[16] Intel Corporation. Flow APIs for hardware offloads, November 2014. Open vSwitch Fall Confernce Talk. `http://openvswitch.org/support/ovscon2014/18/1430-hardware-based-packet-processing.pdf`.

[17] Keon Jang, Sangjin Han, Seungyeop Han, Sue Moon, and KyoungSoo Park. SSLShader: Cheap SSL acceleration with commodity processors. In *8th USENIX Symposium on Networked Systems Design and Implementation*, 2011.

[18] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA efficiently for key-value services. In *ACM Conference on SIGCOMM*, 2014.

[19] Sangman Kim, Seonggu Huh, Xinya Zhang, Yige Hu, Amir Wated, Emmett Witchel, and Mark Silberstein. GPUnet: Networking abstractions for GPU programs. In *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[20] P. Kutch. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. *Intel application note*, 321211–002, January 2011.

[21] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. MICA: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation*, 2014.

[22] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In *USENIX Annual Technical Conference*, 2013.

[23] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15(3):217–252, August 1997.

[24] Daniel Molka, Daniel Hackenberg, and Robert Schöne. Main memory and cache performance of Intel Sandy Bridge and AMD Bulldozer. In *Workshop on Memory Systems Performance and Correctness*, 2014.

[25] Netronome. NFP-6xxx flow processor. `https://netronome.com/product/nfp-6xxx/`.

[26] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. Scale-out NUMA. In *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[27] PCI-SIG. Atomic operations. PCI-SIG Engineering Change Notice, January 2008. `https://www.pcisig.com/specifications/pciexpress/specifications/ECN_Atomic_Ops_080417.pdf`.

[28] PCI-SIG. TLP processing hints. PCI-SIG Engineering Change Notice, September 2008. `https://www.pcisig.com/specifications/pciexpress/specifications/ECN_TPH_11Sept08.pdf`.

[29] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The operating system is the control plane. In *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[30] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *12th ACM Workshop on Hot Topics in Networks*, 2009.

[31] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation*, 2015.

[32] Ian Pratt and Keir Fraser. Arsenic: A user-accessible Gigabit Ethernet interface. In *20th IEEE International Conference on Computer Communications*, 2001.

[33] Kaushik Kumar Ram, Jayaram Mudigonda, Alan L. Cox, Scott Rixner, Parthasarathy Ranganathan, and Jose Renato Santos. sNICh: Efficient last hop networking in the data center. In *6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2010.

[34] RDMA Consortium. Architectural specifications for RDMA over TCP/IP. `http://www.rdmaconsortium.org/`.

[35] Pravin Shinde, Antoine Kaufmann, Timothy Roscoe, and Stefan Kaestle. We need to talk about NICs. In *14th Workshop on Hot Topics in Operating Systems*, 2013.

[36] Weibin Sun and Robert Ricci. Fast and flexible: Parallel packet processing with GPUs and Click. In *9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2013.

[37] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: a user-level network interface for parallel and distributed computing. In *15th ACM Symposium on Operating Systems Principles*, 1995.

[38] N. Zilberman, Y. Audzevich, G.A. Covington, and A.W. Moore. NetFPGA SUME: Toward 100 Gbps as research commodity. *IEEE Micro*, 34(5):32–41, September 2014.