Beyond SmartNICs: Towards a Fully Programmable Cloud

(Invited Paper)

Adrian Caulfield
Microsoft Research
acaulfie@microsoft.com

Paolo Costa
Microsoft Research
pcosta@microsoft.com

Monia Ghobadi
Microsoft Research
mgh@microsoft.com

***Abstract*—FPGA-based SmartNICs and programmable switches have been recently introduced to leverage hardware acceleration and custom pipelines inside the cloud infrastructure. These devices are capable of handling the per-packet processing needs at line rate, including load balancing, encapsulation, congestion management, and security. We argue, however, that the benefits provided by these new devices could extend beyond software-defined networking use cases and they prompt a shift towards a fully programmable cloud, which would enable hardware-software co-design across all layers, ranging from application to hardware and networks. In this paper, we focus on the potential of FPGA-based SmartNICs and programmable switches to realize this vision and illustrate some of the research challenges that need to be addressed to fully unleash its benefits.**

I. Introduction

The continuous growth of cloud applications [1] is driving a steady increase in network infrastructure's bandwidth [2]. However, the compute cycles—measured by the number of CPU cycles required to process each packet—are falling behind the massive acceleration in available network bandwidth [3]. As a result, CPU time is increasingly becoming a contributor to the per-packet latency in high-speed cloud data centers. To make matters more challenging, modern clouds are embracing a fully software-defined network (SDN) and are increasingly expected to perform complex network policies such as regular expression matching and encryption [4]. Such policies drive up the per-packet CPU cycles, which in turn increases the cloud costs and adds unpredictable latency to the cloud services.

Traditionally, this has been addressed by offloading various networking functions to the Network Interface Cards (NICs) such as TCP Segmentation Offload [5] and Generic Receive Offload [6]. However, these techniques are not flexible enough to support complex policies. Techniques such as SR-IOV enable VMs to bypass the hypervisor and send packets directly to the NIC [7]. PCIe Process Address Space ID (PASID) reduces the hardware resource requirements of SR-IOV, enabling it to scale to support containers or even individual processes. But, these techniques bypass the hypervisor, making it hard to enforce SDN-like policies.

Recently, FPGA-based SmartNICs have been introduced as a new platform that enables network operators a flexible environment to offload complex network policies and maintain bandwidth growth in the coming years [4], [8]–[11]. Today's FPGA-based SmartNICs are capable of tracking user-defined state, conducting basic integer arithmetic, and rate limiting at line rate [10]. Moreover, FPGAs organize computation *spatially*, hence data flows through the computation in a pipeline. This minimizes or eliminates latency jitter and provides strong guarantees about throughput. Consequently, the research community has turned its attention towards building platforms and programming languages around FPGA-based SmartNICs [4], [12]–[14].

Most prior work treats the FPGA and the NIC domains separately by either focusing on the FPGA capabilities as a generic device or focusing on the NIC functions. In this paper, we turn our attention into the combined domains and argue that an FPGA-based SmartNIC should be thought of as both a programmable FPGA-based accelerator and a networking device. When combined with recently proposed programmable switches, e.g., [15], [16], this opens up exciting opportunities to rethink the way in which we design and deploy applications and network functions. We argue that we should move away from the traditional strict boundary between network and application functions towards a fully programmable cloud, in which application logic can be distributed across multiple accelerators and network devices.

A fully programmable cloud provides significant benefits to applications that run in it. These benefits include application specific control of network flows, the ability to run code at precisely the right location in the network hierarchy, and direct, low latency access to the network. For example, in a large-scale machine learning workload, a neural network running on a distributed set of SmartNICs would benefit from the direct interface to the network to reduce inference and training latencies. Further, programmable switches running custom flow management code can reduce latency and optimize bandwidth by scheduling flows in an application-specific way, improving efficiency. Finally, the switch could even host a parameter server, directly performing aggregation of the training weights from the SmartNICs below it.

After summarizing the key technology underpinning SmartNICs in Section II, we describe our vision underlying the fully programmable cloud in Section III. Implementing this vision, however, requires solving a number of novel and exciting research questions, which we outline in Section IV.

II. Background

Network accelerators and SmartNICs cover a spectrum from completely CPU-based designs to fully customized hardware. CPU-based programmable NICs provide one or more generic processors on which algorithms can be run. Flow-state and packet-data are brought into the CPU core via load/store instructions and a memory hierarchy, and a sequence of instructions is executed serially to implement the algorithm. Inspecting a single packet may take many instructions, and so often the only way to achieve high throughput is to dedicate multiple cores to the task. Many programmable, CPU-based NICs offer some hardware specialization that optimizes packet data-copying or accelerates common functions like Cyclic Redundancy Check (CRC) calculations or segmentation in their ASICs. Such approaches can help conserve CPU cycles for more irregular algorithms, but increasing line-rates will continue to drive computational demand up.

At the other end of the spectrum, network protocols can be implemented with fully customized hardware, realized as ASICs or FPGAs, with little or no general purpose CPU. Such designs leverage the ability to spatially map the computation they require across an area of silicon. Rather than having a fixed instruction or cycle budget per packet, these NICs have an area budget into which all of the circuitry must fit. Customized hardware typically has little trouble hitting a given throughput target since packet processing can be extensively pipelined and memory access patterns and latencies are more controllable, leading to predictable latencies.

FPGAs and custom chips exploit *spatial* execution. In essence, they take a sequence of instructions, map them into hardware constructs and lay them out in a pipeline on the chip, with each instruction occupying a unique physical area of the chip. Data then streams through the pipeline and all instructions can execute at once. This is generally good for data paths since it provides much higher throughput because each instruction has dedicated resources and is "always running." In contrast, a typical processor uses *temporal* execution, the transpose of the above, where instructions are streamed through a small working set of data. A small number of instructions execute on a slice of data and then more instructions or more data must be swapped in. This model provides great flexibility, but limits throughput, and so works better for complex control paths.

A fully customized architecture could be implemented either as an ASIC or mapped onto a programmable accelerator, such as an FPGA. Both provide spatial computation, but once manufactured, the ASIC can't be changed, so only the latter provides the flexibility to implement new algorithms, fix bugs, and adapt to changing protocols and network requirements. FPGAs provide a unique design point for programmable NICs because they provide many of the benefits of general purpose CPUs in their ability to be reprogrammed and the efficiency and throughput characteristics of fully customized hardware.

Figure 1 illustrates the conceptual integration of the FPGA with the host server in a SmartNIC such as [4]. The CPUs of

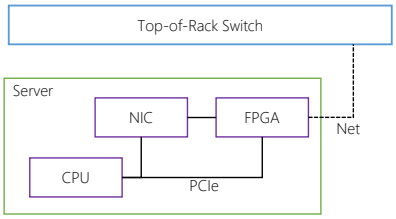

Fig. 1. FPGA NIC integration with data center servers. The FPGA sits inline between the NIC and the top of rack switch, but also has direct PCIe connectivity to the CPU. The network links are typically 40-100 Gbps.

the host system can interact with the FPGA directly using PCIe or by sending packets through the NIC. Unlike conventional server deployments, the NIC is not directly connected to the top-of-rack switch (TOR) and instead is wired to the FPGA card. The FPGA card is the only component connected to the TOR and is responsible for forwarding NIC-to-TOR traffic on behalf of the server.

The key motivation behind this so-called "Bump in the Wire" architecture is to allow the FPGA to perform line-rate computation of in-bound and out-bound network traffic without interrupting the CPUs, thereby saving significant CPU cycles [4] (*network acceleration*). This same architecture lends itself well to accelerate applications for use in an *application acceleration* role as well. The direct PCIe attachment allows software to exchange data at high bandwidth with simple, non-packetized protocols and allows the accelerator to access system memory for large working sets. The architecture has been used in this capacity to great effect, doubling Bing's page ranking throughput while at the same time reducing tail latencies [10]. Rather than relatively simple packet-by-packet processing as in the SmartNIC case, higher-level application processing can implement full microservices directly in the hardware [17]–[19]. Tight integration of the microservice and network effectively allows an extension of the spatial computation across the network.

Traditionally, these two FPGA-based acceleration use cases (application and network) have remained isolated, but the combination of compute accelerator with SmartNIC is powerful, enabling applications to run on and migrate between CPUs and FPGA-based SmartNICs directly at the edge of the network, or even in the network core, to get the best efficiency. We elaborate more on this vision in the next section.

III. A Fully Programmable Cloud

Applications and networks have generally been separated by a strict boundary as dictated by the end-to-end principle [20]. Network switches and interface cards are responsible for executing relatively simple functionality in hardware, e.g., forwarding or queuing packets, while more complex functions, such as congestion control or application logic, are offloaded to the CPU or hardware accelerators located at the network end points. While this principle has served us well in the development of the Internet, it is also the source of several inefficiencies observed in today's data centers. First, ASICs found in NICs and switches are usually bloated with several

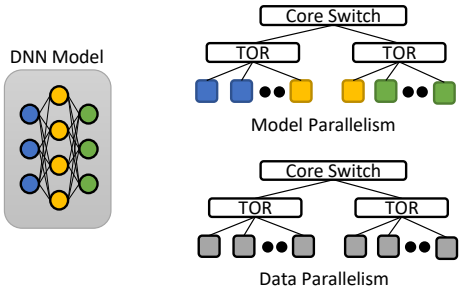

Fig. 2. A DNN model is partitioned across multiple nodes in a model-parallelism case, and the whole model is replicated in the data-parallel case.

features that are not needed and, at the same time, adding new features may require a few years lead time and a tight interaction with hardware manufactures due to the complexity and cost of designing a new chip. Second, black-box congestion control like TCP struggle to achieve high performance and fairness due to the lack of a fine-grain visibility on the network status. Third, applications often experience sub-optimal performance due to a mismatch between network and application routing. Finally, some applications, e.g., coordination services like ZooKeeper [21] or partition/aggregation frameworks such as MapReduce [22], would benefit from the ability to run at vantage points within the network.

The availability of programmable network devices is challenging this status quo. Network designers can now easily customize packet processing and forwarding with almost the same flexibility as software while still retaining hardware speeds. For example, SDN policies can be offloaded from the hypervisor to the NIC [4], new load-balancing protocols could be entirely implemented within the switch ASIC [23], and new features, e.g., header trimming, could be easily deployed to improve end-host congestion control [24].

Albeit very useful, improving the performance of network functions is just the first step. Looking ahead, we argue that the flexibility and programmability offered by network devices has the potential to completely revolutionize the way we deploy and design applications in data centers by allowing the application logic to be distributed across the entire network. The same way in which today an application is split across the CPU and different accelerators (e.g., GPGPUs or FPGAs [19]), we envision that SmartNICs and programmable switches should be seen as just one additional class of accelerators to which critical applications components can be offloaded.

As a concrete example, let's consider a large-scale training of a deep neural network (DNN), spanning multiple hosts. This is typically achieved using either *data-* or *model-parallelism* [25], as shown in Figure 2. In the former, DNN replicas are trained on different servers using different samples of data. After each iteration, their weights are aggregated together and the resulting values are used for the next training iteration. In model-parallelism, instead, a single DNN is distributed across multiple servers (e.g., one layer per server) and servers exchange the intermediate states. In both approaches, a training step consists of computation heavy matrix multiplications, followed by some communication exchanges of the results over the network. The computation-heavy part of the application can be accelerated by offloading it to the FPGA. By having the FPGA next to the NIC, this design drastically improves the IO-intensive part by completely bypassing the network stack. This can be further improved in a fully programmable cloud by taking advantage of the NIC and switch programmability together. For example, leveraging the regularity and long execution time of these applications, we could design a TDMA-based scheme to orchestrate these exchanges so as to minimize network contention and ensure predictable latency, which would drastically improve the performance compared to the traditional black-box approaches like TCP. Further, the aggregation step could be offloaded to the network switch, thus reducing the overall network traffic and end-to-end latency.

While we are already beginning to see some proposals exploiting these new opportunities offered by programmable network devices, e.g., [26]–[29], they are mostly ad hoc solutions, which are hard to generalize. In contrast, we believe that fully unleashing the potential of this vision requires a complete redesign of the software, hardware, and network stack, which in turn poses some stimulating research challenges as we detail in the next section.

IV. Future Research Directions

In this section we identify and discuss key research opportunities for FPGA-based SmartNICs and the programmable cloud, focusing on programming abstractions, resource management, network predictability, and the FPGA hardware itself.

A. Programming Abstractions

A key challenge to realizing the vision outlined in the previous section is to identify the right programming abstractions and tools for developers. Today's SmartNIC platforms are largely programmed using hardware description languages (HDLs) such as Verilog and VHDL. These languages enable tight control of hardware resources but they also bring a barrier to wider adoption as (i) most developers are often unfamiliar to them; (ii) they force to think in terms of a constrained hardware pipeline rather than a high-level algorithm; and (iii) it is difficult to port the code from one platform to another.

To date, many proposals have been made to address this shortcoming. At the high level, these can be grouped into two main classes. On the one hand, there are frameworks such as P4 [30], ClickNP [31], and Emu [32], which focus on providing a packet-based API to process network streams. While these greatly alleviate the effort of deploying network functions and enforcing network policies, their packet-based interface is too low-level for the kind of applications that we have envisioned in Section III. At the other hand of the spectrum, there are several efforts to bring full C/C++ and OpenCL support to these new devices, e.g., [33], [34]. While promising, it is difficult for the state-of-the-art compilers to generate efficient hardware code. This is because there are subtle intricacies of the C programming and memory model,

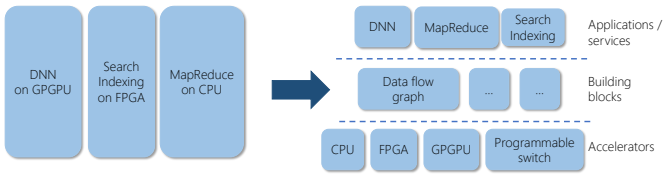

Fig. 3. Programming abstractions: from a vertical to a horizontally integrated stack. Examples are taken from [10], [22], [35].

and a fundamental mismatch between a standard x86 platform (or GPGPU in the case of OpenCL) and these new devices such as the SmartNIC or programmable switches.

In contrast, rather than targeting a general-purpose language we argue that a more suitable approach would be to restrict the focus to the applications of interest and identify the set of common building blocks underpinning them. As shown in Figure 3, the intermediate layer comprising the building blocks decouple the application logic from the underlying implementation of these blocks, shifting from a vertically-integrated stack to an horizontally-integrated one. This is reminiscent of the IP hourglass design where the IP layer decouples the higher-layer protocols from the variety of underlying physical networks, enabling independent innovation and optimizations.

The ultimate goal is to allow application developers to program in a platform-independent fashion, using high-level abstractions and their preferred language, while allowing hardware designers to efficiently implement these building blocks onto a variety of different platforms, e.g., SmartNICs, GPGPUs, and programmable switches, as well as traditional x86 architectures. While a proper investigation of the most suitable building blocks is beyond the scope of this paper, one promising example of building blocks is the data-flow graph abstraction. This allows building different classes of applications with relatively limited effort, e.g., MapReduce, Search [10], and neural networks [29], as they share a similar abstraction. At the same time, the data-flow graph abstraction also matches well with the *spatial* programming model of the FPGA (§ II) and should be easy to port to other accelerators too. Finally, by making the relationships between different component explicit, this abstraction makes it easy for the compiler to reason about and to optimize the resulting code.

B. Resource Management

Another advantage of decoupling the application code from the low-level implementation is that this can enable efficient deployment optimizations at runtime. Given an application, the data center operators can decide the best location to execute the code. We expect that SmartNICs and programmable switches have limited resources so it would be hard to imagine that all user code can be deployed and execute concurrently. Instead, we envision that at runtime, depending on the current resource utilization and the criticality of the task at hand (e.g., core vs. background services or premium vs. standard tenants), the runtime system can decide which applications should be given access to these accelerators. For example, critical machine-learning jobs could be deployed on the SmartNICs with an optimized network stack while non-critical ones could be executed onto the GPUs using standard TCP or RDMA stacks or even run entirely in software.

Since the application code is transparent to the specific hardware technology, we can defer the choice of the platform to the runtime without requiring any effort to the developer. This is in contrast to current approaches in which code that is written for one platform (e.g., Verilog code for SmartNIC or P4 code for the switch) cannot be easily executed on a different platform. Also, by constraining the programmer to use well-defined abstractions with clear semantics, the task of the runtime system is greatly simplified as the application structure and the communication between nodes is explicit rather than opaque as in most of today's applications. For example, based on the expected traffic within the application components (possibly estimated based on prior executions) and the current network utilization, the runtime can decide whether it makes sense to host within the same rack (high bandwidth at the cost of fragmentation) or spread them around (lower bandwidth but more efficient packing). Similarly, it can decide whether a service would benefit from running on a programmable switch or on a SmartNIC.

Extracting the best performance out of the combination of these programmable parts requires careful placement of instances to account for network locality, tools for management and scheduling workloads, and handling of failures. This shares many challenges with other job and resource scheduling frameworks but it also introduces new ones. Service instances should be located close to their dependencies and dependants, minimizing network latency and making the best use of bandwidth. Resources are constrained on these platforms, hence schedulers must make prioritization decisions about what jobs to run, or what fraction of the FPGA is available for a given service. As an example, a microservice could run on a top-of-rack switch and perform parameter aggregation for a collection of nodes under it. The service can allocate resources on the switch proportionate to the number of ports running the DNN operations.

One of the key design decisions is the trade-off between ephemeral and stateful services. Ephemeral services, or those which can be quickly re-initialized with a small dataset and torn down without saving state, make ideal hardware microservices. Their ephemeral nature allows a resource manager to rapidly scale up or down the number of instances to account for varying load or to relocate services to optimize network utilization. A DNN inferencing engine is a good example of this type of service. Instances can be brought online, initialized with a set of model weights and then begin serving. Stateless services may call out to stateful ones to query databases or access storage. The direct network connectivity keeps the latency of these external calls low.

Beside investigating new scheduling policies, a key challenge is also how to efficiently share a single resource among different applications. This is in stark contrast to today's practice of assigning each resource exclusively. One solution

could be to use time division multiplexing (TDM) and dedicate the entire FPGA to one application at a time. Another approach is to use space division multiplexing (SDM) and share the FPGA across multiple microservices. TDM requires including support for saving and restoring application state, equivalent to saving process state during a CPU context switch. FPGAs do not inherently support this, but it could be implemented using reconfigurable logic. Partial reconfiguration allows reprogramming a pre-defined section of the FPGA without interfering with other running logic. This technique can be used to effectively combine TDM and SDM, but many challenges remain for its effective adoption. For example, logic must be compiled targeting a particular region while the resource allocations are fixed. Ultimately, the goal is to create a virtualization layer for SmartNICs and programmable switches that would seamlessly allow sharing resources among different applications without compromising on security and performance isolation.

C. Network predictability

Besides improving the performance of next-generation applications, a programmable cloud infrastructure could also improve the performance of existing cloud applications by allowing a custom control of the network. This opens up interesting research opportunities to enable elements in the network to be programmed to enforce latency bounds. Traditionally, one of the main challenges of providing latency Service Level Agreements (SLA) has lied in the lack of a programmable infrastructure. FPGA-based SmartNICs and programmable switches give us an opportunity to solve this problem head-on, as it is possible to track connection state on the FPGA and perform rate limiting and traffic management of a large number of queues.

The larger goal is to offload flow scheduling, congestion control, and other factors that affect application's latency in the FPGA-based SmartNICs and programmable switches. This, in turn, enables the network controller to reason about the latency of packets as well as to prioritize packets that are close to violating their latency SLAs. Doing so has the potential of enabling cloud providers to provide latency bounds for applications and cloud customers to purchase latency according to their needs.

The first step to realize network predictability is to find the right abstractions that map network functions into hardware capabilities (such as rate limiters and priority queues) in a programmable fashion. Using these abstractions, the next step is to build scalable network functions on the FPGA by determining proper control loops for marking traffic (such as ECN marks) and for reacting to signals from the network (such as ACK packets). Approaches such as PIFO [36] and pFabric [37] are great examples of network abstractions that can be implemented in hardware. However, these proposals have focused on programmable network switches thus far.

Another advantage of hardware-based services is the ability to understand the data pipeline and reason about delays in each hardware module. This, in turn, enables network operators to profile and predict the runtime latency of programs that run in hardware. Further, the compiler can automatically find possible microservice candidates in the code and provide the predicted latency for that microservice. For example, consider an application with a heavy duty matrix multiplication in its code, the compiler can tag hardware acceleration for the matrix multiplication function. The network scheduler then observes the state of the network, along with application's SLA on latency and decides on the distribution of tasks across available resources. Doing so bridges the traditional separation between applications and networks and opens doors to have applications with end-to-end latency bounds.

D. FPGA Hardware

The FPGAs in use today for SmartNIC and data center acceleration tasks are descendants of hardware designed for different problem domains. Large FPGAs have generally found use in low volume signal processing and ASIC prototyping scenarios, while small ones are commonly used as "glue-logic" between other components. This heritage gives FPGAs a solid grounding to tackle line-rate network and acceleration tasks because of their ability to map computation spatially. Advances in FPGA logic resources, memory capacity, and clock rates have made the devices attractive targets for data center acceleration tasks. However, only the latest FPGA generation have been adopted in data centers, hence ample opportunity remains to evolve their hardware architecture to better suit this use case. One example could be incorporating associative memories (CAMs) to aid in implementing fast searches, or narrow bit-width multipliers for neural network computations.

With the end of Moore's law, chip designers and computer architects must be more selective in deciding which functions are included in a chip. A good example of this cost trade-off is the split between the NIC and FPGA in the SmartNIC architecture (Figure 1). The separation of NIC and FPGA in the architecture was a conscious choice since the whole design could be implemented in soft-logic on the FPGA if desired. However, the protocol offload, host interfaces, and other functionality of the NIC remain constant, and so it is "cheaper" (from a transistor count stand-point) to harden that functionality. The PCIe and network bandwidths do not change over time in a given system. This reserves the relatively expensive reconfigurable logic for parts of the design that will need to change frequently.

Over time, some parts of the FPGA-based logic will stabilize enough and be universally useful enough to consider hardening. The FPGAs of tomorrow will effectively be hybrids – part ASIC, part reconfigurable fabric. In fact, this trend has already started in today's FPGAs. PCI Express, network MAC and PHY layers, memory controllers and math units already exist as "hardened" sub-components in the FPGA fabric. Often the data path is hardened with small, likely to change control paths left to the reconfigurable logic. These devices start to look a lot like system on chips (SoCs) or coarse-grained reconfigurable architectures. Some FPGAs even have CPUs built in and tightly connected to the fabric. An ongoing and

iterative research process will be required to identify the best set of building blocks and kernels to harden and the correct abstractions to expose them.

In the context of the DNN example and vision, the rapid pace of evolution in DNN algorithms means its currently better to spend the transistors on a flexible FPGA fabric so the algorithms and computation can rapidly evolve as needed. Eventually, if DNNs remain popular and a stable algorithm emerges, it may be hardened.

While new generations of FPGAs will continue to evolve and incorporate new hardened functionality, there may be transformational shifts coming in how reprogrammable chips are designed and used. Support for rapid context switching between two or more circuits on an FPGA would allow better virtualization but requires saving and restoring the state of the chip. Finally, non-traditional FPGA implementation technologies such as those using asynchronous logic [38] may provide better performance, logic density, or power usage, but are not widely in use yet.

V. Conclusion

The continuous growth of modern cloud networking has driven huge increases in the per-packet processing complexity and performance requirements for workloads. However, CPU based approaches have reached the throughput limits of generic processor architectures. The recent introduction of FPGA-based SmartNICs and programmable switches have provided a flexible and scalable platform with promising results on packet-level processing. The FPGA-based SmartNIC architecture is suitable for both packet processing as well as application-level acceleration, and in combination, presents an opportunity to deploy full applications that can tightly integrate with, and extend spatial computation across, the network. In this paper, we lay out a number of research directions covering programming abstractions, ability to share and schedule tasks throughout the network, and future directions for hardware improvements.

Acknowledgment

The authors would like to thank Doug Burger, Eric Chung and the Catapult and Project BrainWave teams at Microsoft.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 20162021," https://bit.ly/2wmdZJb.
[2] Y. Sverdlik, "Private Data Exchange is Outpacing Internet's Growth," https://bit.ly/2wJBk8N.
[3] F. Kruger, "CPU Bandwidth The Worrisome 2020 Trend," https://bit.ly/2rKhjtL.
[4] D. Firestone *et al.*, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *NSDI'18*, 2018.
[5] G. W. Connery, W. P. Sherer, G. Jaszewski, and J. S. Binder, "Offload of TCP segmentation to a smart adapter," 1999, uS Patent 5,937,169.
[6] "net: Generic receive offload," https://lwn.net/Articles/358910/, 2008.
[7] D. Firestone, "VFP: A Virtual Switch Platform for Host SDN in the Public Cloud." in *NSDI*, 2017.
[8] "Netronome SmartNICs," https://bit.ly/2iFcoEs.
[9] Will Chu, "Intelligent networks by Cavium," http://www.cavium.com/newsevents_Caviumnetworks_CoredgeNetworks.html.
[10] A. Putnam *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *ISCA*, 2014.
[11] "Mellanox Innova-2 Flex Programmable Network Adapter," https://bit.ly/2rIIZim.
[12] M. T. Arashloo, M. Ghobadi, J. Rexford, and D. Walker, "Hotcocoa: Hardware congestion control abstractions," in *HotNets*, 2017.
[13] K. Olukotun, "Scaling Machine Learning Performance with Moore's Law," https://bit.ly/2rIHiS2.
[14] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, pp. 85–98.
[15] "SONiC, the network innovation powerhouse behind Azure," https://bit.ly/2GNiy1z.
[16] "Barefoot Technology," https://www.barefootnetworks.com/technology/.
[17] D. Burger, "A New Era of Hardware Microservices in the Cloud," https://bit.ly/2Ir9nE8.
[18] M. Branscombe, "FPGAs and the New Era of Cloud-based Hardware Microservices," https://thenewstack.io/developers-fpgas-cloud/.
[19] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture." IEEE Computer Society, October 2016.
[20] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end Arguments in System Design," *TOCS*, vol. 2, no. 4, 1984.
[21] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems," in *ATC*, 2010.
[22] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI*, 2004.
[23] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs," in *SIGCOMM*, 2017.
[24] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance," in *SIGCOMM*, 2017.
[25] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large Scale Distributed Deep Networks," in *NIPS*, 2012.
[26] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres," in *CoNEXT*, 2014.
[27] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "NetChain: Scale-Free Sub-RTT Coordination," in *NSDI*, 2018.
[28] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang, "KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC," in *SOSP*, 2017.
[29] E. Chung *et al.*, "Accelerating Persistent Neural Networks at Datacenter Scale," in *HotChips*, 2017.
[30] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *CCR'14*.
[31] B. Li, K. Tan, L. Luo, R. Luo, Y. Peng, N. Xu, Y. Xiong, and P. Cheng, "Clicknp: Highly flexible and high-performance network processing with reconfigurable hardware," in *SIGCOMM*, 2016.
[32] N. Sultana, S. Galea, D. Greaves, M. Wojcik, J. Shipton, R. G. Clegg, L. Mai, P. Bressana, R. Soule, R. Mortier, P. Costa, P. Pietzuch, J. Crowcroft, A. W. Moore, and N. Zilberman, "Emu: Rapid Prototyping of Networking Services," in *ATC*, 2017.
[33] "Intel HLS Compiler," https://bit.ly/2ImqR8s.
[34] "Vivado High-Level Synthesis," https://bit.ly/2vcBwee.
[35] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The Microsoft 2017 Conversational Speech Recognition System," Microsoft Research, Tech. Rep., August 2017.
[36] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable packet scheduling at line rate," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 44–57.
[37] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446.
[38] C. LaFrieda, B. Hill, and R. Manohar, "An Asynchronous FPGA with Two-Phase Enable-Scaled Routing," in *ASYNC*, 2010.