Report from Dagstuhl Seminar 24291

Programmable Host Networking

Gianni Antichi*[1], Katerina Argyraki*[2], Aurojit Panda*[3], and Justine Sherry*[4]

1 **Politecnico di Milano, Italy.** gianni.antichi@polimi.it
2 **EPFL, Switzerland.** katerina.argyraki@epfl.ch
3 **New York University, USA.** apanda@cs.nyu.edu
4 **Carnegie Mellon University, USA.** sherry@cs.cmu.edu

Abstract

Increasingly communication software is being offloaded to specialized hardware accelerators and into the OS kernel. In most cases this is because offloading is supposed to improve network utilization and reduce costs. However, designing good offloads is challenging, often requiring architectural changes to both software and hardware. But there is little agreement on the form of these changes, and this both increases the true cost of building and deploying offloads and the complexity of doing research on accelerators. This Dagstuhl Seminar aimed to provide a forum to talk about experiences with building and deploying accelerator platforms to address this concern.

Seminar July 14–19, 2024 – https://www.dagstuhl.de/24291
2012 ACM Subject Classification Hardware → Buses and high-speed links; Computer systems organization → Architectures
Keywords and phrases Networking, Accelerators, Interconnects
Digital Object Identifier 10.4230/DagRep.14.7.35

1 Executive Summary

Gianni Antichi
Katerina Argyraki
Aurojit Panda
Justine Sherry

Over the past two-decades network link speeds have grown faster than processor clock speeds. Consequently, the software and hardware that applications use to communicate is often the bottleneck. Eliminating this bottleneck requires redesigning the software network stack and hardware NICs that applications use to send or receive messages. Until now changing either was challenging, but recent changes to the software ecosystem have made it possible to prototype, and then deploy new network stack and NIC designs. On the software side, the addition of eBPF to the Linux and FreeBSD kernels have made it easier to modify the network stack *without* needing to change the kernel, while on the hardware side, the availability and adoption of programmable SmartNICs such as Xilinx Alveo and Intel Agilex allows us to now change both NIC design and the NIC-OS interface without needing to upgrade or change server hardware.

However, at present there is no consensus on how NIC-OS-Userspace should be architected, what features they should provide, or how they should be implemented. This is

* Editor / Organizer

because answering these questions requires combining techniques and ideas from several sub-disciplines including networking, systems, computer architecture, programming languages, and compiler design. This seminar's goal was to bring together academics and industry practitioners who had expertise in these areas and discuss how to best architect and build platforms that enable accelerator use.

We structured the seminar around four core questions:

(a) What hardware accelerators are used in practice, and how?
(b) What interfaces do current OS accelerator frameworks provide, what are their limitations, and how are they used?
(c) How do hardware accelerators communicate and coordinate with processors?
(d) What tools do we have available to reason about the performance of both hardware and software accelerators?

The choice of questions was informed by the set of attendees, and by the organizers experiences in this area. Surprisingly we found that there was significant disagreement on the desirability of both hardware and software accelerators, with many of the attendees arguing that they might be a temporary measure while we figure out what future servers look like. This is a different conclusion than what one would arrive at by looking at recent papers in the systems, networking, and architecture communities, and from current industry deployments. Our report provides a more detailed account of the seminar's discussions.

2 Table of Contents

3 Seminar Structure

We structured the seminar around a series of tutorials that set the agenda for the day. On Monday, Tuesday and Thursday, we had two to three tutorials, which were followed by shorter talks or discussions on topics raised during the tutorial. Wednesday was a half-day because of the traditional Dagstuhl outing, and was thus anchored by only one tutorial, and many of the participants had to leave early on Friday, and we organized a few talks on ongoing research rather than following the usual structure. Consequently, we have organized this report by day: we briefly describe the tutorials and the takeaways from them, and then describe the other activities on that day.

4 Monday: How does industry use offloads?

4.1 Tutorial 1: Groq's FPGA Based ML Cluster

Satnam Singh (Groq - Mountain View, US)

The first tutorial of the workshop was by Satnam Singh who works at Groq [1], a startup that developed a custom ML inference cluster architecture, and implements this architecture using FPGAs. Using FPGA improves flexibility: Satnam explained that Groq can choose how to implement operations, how to connect them, etc. However, it should come at a performance cost because GPU clock speeds are significantly higher than FPGAs.

Satnam explained that this is a case where the flexibility of programmable logic is a bigger help than any loss in performance: like other ML cluster, Groq distributed computation across the cluster, but unlike with GPUs they know at a cycle granularity when each computation will complete. This allows them to design clusters where explicit coordination (through barriers) is not required, and where no cycles are wasted waiting for data to arrive. Both in combination allow them to achieve better performance than GPU clusters.

The group's takeaway from this talk was that programmable logic is useful, even if it comes at the cost of clock speeds or other low-level performance metrics.

References

1 Groq. The Groq LPU Inference Engine.
https://wow.groq.com/lpu-inference-engine/.

4.2 Tutorial 2: How do customers use Xilinx and Pensando SmartNICs

Mario Baldi (AMD - Sunnyvale, US)

Our next tutorial was by Mario Baldi, who talked about AMD's SmartNIC offerings (generally branded under Xilinx and Pensando), and how hyperscalers, e.g., Microsoft Azure, were using them. Mario's core takeaway was that SmartNICs are only useful in two scenarios: (a) there is uncertainty about what communication functionality is required; or (b) there

is a desire to prototype new functionality. However, Mario argues that the first should be resolved in time because desired functionality tends to not change very often, and he expects that in the next few years, each cloud provider will be in a position to just harden this into ASICs. He also argued that while the second use case is of interest to academia and research, it really does not apply to production use cases. Thus, somewhat unexpectedly he said that he expects programmable SmartNICs will no longer be in widespread use in a few years. This proclamation led to one of the big discussions (§4.4) later in the day.

4.3 Tutorial 3: SimBricks or How to Prototype Hardware Changes Cheaply

Antoine Kaufmann (MPI-SWS - Saarbrücken, DE) and Jialin Li (National University of Singapore, SG)

Our final tutorial for the day was led by Antoine Kaufmann and Jialin Li, and focused on how to use SimBricks [1] to evaluate changes to computer hardware. The goal of this tutorial was to provide attendees a prototyping tool for evaluating proposals discussed during the seminar. Antoine and Jialin walked through the steps required to get started with SimBricks, how to incorporate VHDL designs, and how to run applications on SimBricks.

References

1 Hejing Li, Jialin Li, and Antoine Kaufmann. SimBricks: End-to-end network system evaluation with modular simulation. In *SIGCOMM*, 2022.

4.4 Discussion: Why SmartNICs?

Our first discussion of the seminar was prompted by Mario's statement (§4.2) that he did not think SmartNICs would be used (within datacenters) within a few years. The discussion was seeded by comments from Mario, Boris Pismenny (Nvidia and EPFL), Dan Tsafrir (Technion), and Mark Silberstein (Technion). Mario and Boris drew upon customer experience (through their respective companies), while Dan and Mark drew upon their prior research to address this question.

Boris, Dan and Mark believed that we were going to continue to have SmartNICs. They argued that this was for two reasons: First, Mario's analysis had missed an important use case for SmartNIC, which is that they provided a way to run computation without fears of side-channel leaks because they did not share resources with the processor; and second, they observed that processing on the SmartNIC was less expensive from a cloud provider's perspective, because SmartNIC cores could not generally be sold. The discussion then centered around understanding these reasons: why are SmartNIC cores cheaper (their seem to be fewer of them in a server), and are SmartNICs really the right way to avoid side channels. While we did not arrive at a definitive conclusion, this discussion also brought up an additional discussion question: what is a SmartNIC, and what programming abstractions should we use when reasoning about their capabilities.

5 Tuesday: Offloads in software

5.1 Tutorial 1: NIC—software interfaces

Boris Pismenny (EPFL - Lausanne, CH)

In this tutorial, Boris provided his perspective on how to rethink the interface between NICs and CPU for high-performance processing [1, 2, 3].

The starting point was that network software interface are evolving with a number of technology trends and NIC upgrades are more cost-effective relative to CPU upgrades. One insight was that NIC offloads are mostly free. These together motivate to look at offloading CPU functionality to the NIC. Boris showed two trends about CPUs available from Intel, AMD, and ARM vendors: (1) CPU I/O speeds are growing faster than CPU core number; and (2) CPU I/O speeds are growing faster than CPU memory bandwidth. These indicate that cores are increasingly unable to process data at line-rate and that memory bandwidth isn't sufficient to store all I/O data, making NIC offloads and other ways the NIC can improve CPU efficiency even more important.

This lead to a discussion on how several software techniques can optimize performance: batching at different layers of the I/O stack; zerocopy on receive, transmit, within applications, and by using device memory. The tutorial provided insights on current research that tackles memory bandwidth bottlenecks by backpressure on cores and the network, and research that fits the workload to the last-level cache to benefit from DDIO.

Finally, Boris discussed several NIC hardware techniques to deal with multi-core CPUs that introduce challenges on fair and efficient packet scheduling on transmit and receive and also advances in NIC SRIOV support that enable matching paravirtualized NIC functionality while gaining the performance benefits of SRIOV.

References

1 Boris Pismenny, Adam Morrison, and Dan Tsafrir. ShRing: Networking with shared receive rings. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2023.

2 Boris Pismenny, Liran Liss, Adam Morrison, and Dan Tsafrir. The Benefits of General-Purpose on-NIC Memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, 2022.

3 Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafrir. Autonomous nic offloads. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, 2021.

5.2 Tutorial 2: Networking with eBPF

Paul Chaigon (Isovalent - Rennes, FR)

Paul's tutorial focused on providing an overview of eBPF and its use in the Linux kernel. He started by presenting program characteristics: in particular focusing on the fact that the programs need to be checked by a verifier before loading to limit buggy behavior and crashes. The use of a verifier, rather than a sandbox, minimizes runtime overheads but requires that the kernel provide helper functions and collections.

Paul then talked about usecases he has seen so far, including Cilium [1], a proxy developed at Isovalent that was intended to replace `netfilter` and `kubeproxy`; Katran [2], a loadbalancer developed at Meta; XDP [3] based DDoS mitigation software developed at Cloudflare, Cilium and Meta; and programs that bypass the network stack including ones to redirect messages between sockets (`sockmap`).

Finally, Paul spoke a bit about planned improvements that are designed to make it easier to write eBPF programs: changes to the verifier are likely to allow some unbounded loops and function calls; dynamic memory allocation is now allowed; eBPF programs can use timers; and implement their own datastructures. He also pointed out that these changes affect the entire eBPF ecosystem: we are likely to see new libraries that implement eBPF native data structures; and the use of more flexible locks within eBPF programs.

Much of the discussion after the tutorial was focused on whether eBPF in the kernel was a good thing, or was just repeating mistakes from the past.

References

1 Liz Rice. Cilium: How ebpf streamlines the service mesh. `https://thenewstack.io/how-ebpf-streamlines-the-service-mesh/`, 2021.
2 Nikita Shirokov and Ranjeeth Dasineni. Katran, a scalable network load balancer. `https://engineering.fb.com/2018/05/22/open-source/open-sourcing-katran-a-scalable-network-load-balancer/`, 2018.
3 Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: fast programmable packet processing in the operating system kernel. In *International Conference on Emerging Networking EXperiments and Technologies (CoNExT)*. ACM, 2018.

5.3 Tutorial 3: Performance Interfaces

Rishabh Iyer (UC Berkeley, US)

Our last tutorial focused on performance interfaces. Most software comes with semantic interfaces, including code documentation, header files, and specifications, that are necessary for programmers (or users) to integrate and use them. However, software generally does not come with any performance interfaces, making it hard to reason about how library or software use affects performance. Rishabh presented work from his PhD disertation [1] on developing abstractions that allow programs to provide performance interfaces. In developing these abstractions, Rishabh had to answer three core questions: how to represent

performance interfaces; how to generate them for software; and how to compose them to reason about the performance of a larger system? Furthermore, answers to these questions affect the accuracy and utility of the interfaces themselves. The discussion centered around the choices made, and alternatives available.

References

1 Rishabh Ramesh Iyer. Latency interfaces for systems code. Technical report, EPFL, 2023.

6 Wednesday: Enabling hardware research

6.1 Enzian

Timothy Roscoe (ETH Zürich, CH)

Timothy Roscoe presented lessons learned while building Enzian [1], an open platform designed to make it easier to experiment with hybrid computing platforms that include CPU cores, FPGAs and various accelerators. Enzian has been used by multiple projects, and it allows a great degree of flexibility. However, this flexibility has required the Enzian team to often eschew the use of off-the-shelf commercial parts, and instead design new parts: for example, Enzian required the development of a new server board, new baseboard management component, a new interconnect between CPU and FPGA, etc. These increased costs and time to development, but have been crucial to Enzian's success. Mothy's talk discussed the history of this development, and ongoing challenges with maintaining and producing new instances of this hardware.

References

1 David Cock, Abishek Ramdas, Daniel Schwyn, Michael Giardino, Adam Turowski, Zhenhao He, Nora Hossle, Dario Korolija, Melissa Licciardello, Kristina Martsenko, et al. Enzian: an open, general, CPU/FPGA platform for systems software research. In *ASPLOS*, 2022.

6.2 How should accelerators communicate?

Hugo Sadok (Carnegie Mellon University - Pittsburgh, US)

Traditionally CPUs were the only compute platform and peripheral devices simply allowed the CPU to communicate with the outside world. Today, however, peripheral devices are increasingly more powerful and can often work fully independently from the CPU. Yet, the way devices communicate today still assumes that the CPU is the only compute unit. Today devices communicate using a shared memory abstraction that allows devices to read and write data to main memory allowing them to interact efficiently with the CPU. Yet, the shared memory abstraction is inefficient when we need devices to communicate directly within the host. Shared memory forces the CPU into the datapath even when accelerators need to talk to one another. This prevents applications from efficiently using multiple accelerators chained together.

In this talk/discussion I invite researchers to think of what needs to be changed in the way devices communicate inside the host in order to enable a future with more specialized compute platforms. I will raise the following questions, some of which we have proposed solutions and some that we do not: Which communication abstractions are better suited to enable direct communication among devices? What applications can benefit from such new abstractions? What changes should be made to the host interconnect architecture to enable direct communication among devices? Do new abstractions also enable different kinds of accelerators? How should application developers leverage chains of heterogeneous accelerators without sacrificing portability? How to properly manage accelerators when using these new communication abstractions?

6.3 WASM Instead of eBPF?

Gábor Rétvári (Budapest University of Technology & Economics, HU)

In his tutorial, Paul (§5.2) described how eBPF provides kernel extensibility, but its verifier imposes some limits on how programs are written. Gabor asked if should we consider alternatives, e.g., WASM. He pointed out that there is already an initial implementation of WASM in Linux [1], and we mainly need to figure out if it works and if it is better than eBPF. Gabor provided some initial arguments on why we should consiuder using WASM.

References

1 Camblet Driver. `https://github.com/cisco-open/camblet-driver`.

6.4 Enabling System Innovation of Optical Data Center Networks with an Open Framework

Yiting Xia (MPI für Informatik - Saarbrücken, DE)

Optical data center networks (DCNs) are revolutionizing the infrastructure design in the cloud. Despite their great success, today's optical DCN architectures operate as closed ecosystems, locking in system solutions with the underlying optical hardware. This talk presents an open framework that decouples software systems from optical hardware, enabling independent evolution. The framework facilitates the evaluation of software proposals on various hardware architectures and advances the adoption of optical DCNs by substantially simplifying their development. It also makes numerous research topics in optical DCNs more accessible to system and networking researchers, such as time synchronization, routing, and transport designs.

6.5 Metrics Around Energy Usage in Programmable Networked Systems

Deepti Raghavan (Brown University - Providence, US)

Systems that use network offloads are usually motivated by performance efficiency: offloading tasks to a programmable device will save cycles on the CPU host and will increase overall system performance. It is also clear what metrics can be used to measure performance: for example, we can either profile cycles, measure throughput, or measure latency when responding to end-to-end requests on the CPU host. However, would deploying the network offload for a particular task always be the most energy efficient option? What are the ways that we can reason about this or measure this when building systems? In this talk, I would like to discuss ideas around measuring and optimizing for energy efficiency when using network offloads.

6.6 We don't need drivers anymore, translation is the SmartNIC's job

Tom Barbette (UC Louvain, BE)

With a budget of 300 cycles per packet, NFV scenarios and network intensive applications necessitate efficient packet exchange mechanisms between NICs and CPUs where even the latest kernel by-pass techniques, such as DPDK, can eat up a third of the budget. The talk explores the potential to tailor packet descriptors for different applications and different buffering model. We then offload the driver's work from the datapath so the application receives the packets precisely as it needs them. We propose a prototype implementation on NVIDIA Bluefield 3 SoC and explore the limitations of the acceleration mechanisms that can be leveraged on the BlueField 3. Evaluated in multiple NFV scenarios, we show an improvement of 75% more traffic under the same constraints as state-of-the-art solutions.

7 Thursday: Interconnects

7.1 Tutorial 1: PCIe

Rolf Neugebaue (Amazon - Cambridge, GB)

In this tutorial, Rolf provided an overview of PCIe, the de-facto I/O interconnect in contemporary servers. PCIe uses a high-speed serial interconnect based on a point-to-point topology consisting of several serial links (or lanes) between endpoints. It is a protocol with three layers: physical, data link and transaction. While the data link layer (DLL) implements error correction, flow control and acknowledgments, the transaction layer turns

user application data, or completion data, into PCIe transactions using Transaction Layer Packets (TLPs).

After a brief introduction, he delved in the overheads associated to PCIe trasactions taking as a reference point a 40 Gb/s NIC with a PCIe Gen 3 interface with 8 lanes. Here, each lane offers 8 GT/s (Giga Transactions per second) using a 128b/130b encoding, resulting in 8×7.87 Gb/s = 62.96 Gb/s at the physical layer. The DLL adds around 8–10% of overheads due to flow control and acknowledgment messages, leaving around 57.88 Gb/s available at the TLP layer. For each transaction, the physical layer adds 2B of framing and the DLL adds a 6B header. Apart from transferring the packet data itself, a NIC also has to read TX and freelist descriptors, write back RX (and sometimes TX) descriptors and generate interrupts. Device drivers also have to read and update queue pointers on the device. All these interactions are PCIe transactions consuming additional PCIe bandwidth.

The tutorial then moved to specific features of the PCIe such as the IOMMU, which is in modern systems interposed in the data path between a PCIe device and the host. The IOMMU performs address-translation for addresses present in PCIe transactions and utilizes an internal Transaction Lookaside Buffer (TLB) as a cache for translated addresses. On a TLB miss, the IOMMU must perform a full page table walk, which may delay the transaction and thus may increase latency and impact throughput. The problem highlighted is that this can introduce introduce variance as transactions are now depending on the temporal state of caches, IOMMU TLB and the characteristics of the CPU interconnects.

The tutorial concluded with a discussion of scale-up interconnect technologies for machine learning clusters with a specific focus on NVLink. NVLink is a high-speed, wire-based, point-to-point connection technology that allows GPUs to communicate directly with each other within a server. It's a faster alternative to traditional PCIe-based solutions and is used to scale memory and performance for computing demanding workloads.

7.2 Tutorial 2: CXL

Emmanuel Amaro (VMware - Palo Alto, US)

In this tutorial, Emmanuel provided an overview of Compute Express Link (CXL), an open standard interconnect for low latency, high throughput I/O designed to interconnect hosts, accelerators, and memory devices. CXL uses the ubiquitous PCIe electrical interface and defines three protocols: CXL.io, CXL.cache, and CXL.mem. CXL.io provides I/O semantics and is implemented by all CXL devices because it is used by the control path. CXL.cache implements a cache synchronization protocol for device and CPU caches, and CXL.mem provides a transactional interface between CPUs and CXL memory devices.

The tutorial touched upon the three device-types defined by CXL (i.e., Type-1 and Type-2 which are accelerators and Type-3 which are memory devices) and discussed the main motivation behind this new standard interconnect, taking as example memory disaggregation. Many prior research efforts, indeed, has shown the potential of memory disaggregation with high-performance network fabrics. These efforts rely on software memory disaggregation: software inititates requests to access disaggregated memory and acknowledges completions. For instance, using RDMA, application libraries or the OS must post memory access requests to network queues; the NIC then adds completions to completion queues, which software

drains. This process is slow and poorly aligned with CPU architectural features. Further, this problem is common to most network transports, including TCP, where CPU overheads are even larger, exacerbating the performance impact of software memory disaggregation. By integrating CXL ports directly into CPUs, processors can directly address disaggregated memory, enabling CPU architectural features like caching, prefetching, pipelining, and speculative execution. As a result, hardware memory disaggregation reduces CPU overheads, lowers latency, and increases throughput compared to previous software approaches.

The tutorial then moved on early performance results presented by previous research papers alongside new research directions opened by this standard. In particular, Emmanuel first focussed on the question if it is possible to build data structures that provide consistency atop incoherent CXL.mem and took as example in-memory storage for data flow computation. Then, he discussed about how to enable resource composability by providing better efficiency and flexibility on top of a CXL interconnect.

7.3 Affordances for Accessible Application Programming

Akshay Narayan (Brown University - Providence, US)

There are currently two ways to write applications: the "traditional" way, which uses familiar APIs, and the "hard" way, in which these APIs are discarded in favor of lower-level ones with more control over factors that impact performance. I argue that neither of these ways is easy, and this leaves a lot of applications behind, causing programmers to reach for traditional APIs at the expense of benefiting from advances in networking technology.

7.4 Workload-aware in-network crypto scheduler for FPGA NICs

Rinku Shah (IIIT - New Dehli, IN)

Offloading compute-intensive workloads is a routine for the hyperscalers, and cryptographic primitives (e.g., encryption and decryption) are one of the important offload candidates. Current approaches use various solutions such as on-CPU accelerators, off-network-path fixed-function ASICs, and reconfigurable FPGA-based accelerators. However, these solutions mainly focus on a single cryptographic algorithm, while real-world workloads often involve multiple algorithms. For instance, a TLS data path connection uses algorithms such as AES-GCM, AES-CTR, AES-CCM, and Chacha for (de)encryption. FPGA-based accelerators primarily focus on performance or power efficiency, but non-parallelizable workloads see negligible gains via performance accelerators, leading to a waste of hardware resources and power. We propose to design workload-aware cryptographic accelerator primitives for FPGA NIC, and a workload-aware scheduler. We are working on addressing challenges such as work-preservation, HOL blocking and performance isolation for tenants and would highly appreciate your inputs and feedback.

7.5 To TEE or not to TEE?

Mark Siblerstein (Technion - Haifa, IL)

Recently there have been a few designs that propose providing Trusted Execution in SmartNICs. Are there enough use cases for this? I posit that the problem to be solved is either way deeper than a TEE, or it is so niche that it is not that important in practice.

7.6 Discussion: Interconnect Throwdown

We had started the day hearing about two different interconnect technologies: PCIe and CXL. Furthermore, Mothy's talk on Enzian (§6.1) had indicated that current interconnects might be holding us back. To resolve this, we organized a debate about what is necessary and desirable for interconnects within the server. We had Rolf, Emmanuel, Mothy, and Satnam up on stage, and had each of them talk about interconnect differences and efficiencies. The core message was that PCIe and CXL have high-overheads, but these might be necessary to support the range of use-cases that they are currently employed for.

8 Friday: The last day

8.1 Incorporating asynchronous programming in network stack

Sue Moon (KAIST - Daejeon, KR)

In the past few decades, networking systems and frameworks have evolved to address modular design, reusability, and performance. Network I/O is asynchronous by nature, yet most frameworks rely on event-driven programming with callbacks. While event-driven programming ensures high performance, it compromises reusability due to application-specific dependencies among callback functions. The async/await paradigm is gaining momentum as a general abstraction for asynchronous programming by encapsulating asynchronous behaviors into application-independent combinators. As a result, a program's asynchronous behavior is composed from few primitive building blocks. However, the async/await paradigm has had limited exposure in networking systems and frameworks due to their high-throughput, low-latency requirements.

We present AsyncNet, a novel networking system framework that adopts the `async/await` paradigm to meet the high- throughput, low-latency requirements of network systems. AsyncNet addresses three major challenges while adopting the combinator pattern of the async/await paradigm. First, network systems eagerly use polling to eliminate context-switching overhead; however, existing schedulers cause imbalanced scheduling between polling and non-polling tasks. We distinguish between the two types of tasks on separate run queues and provide a tailored scheduling algorithm. Second, techniques for high-throughput networking, such as direct memory access, zero-copy API, and timer management, were not encapsulated within a framework, forcing programmers to embed

such techniques along with the protocol logic. We embed these techniques as part of our framework. With the fine declaration of access patterns, our framework automatically determines whether to make a copy or not. Finally, some unique data movement patterns of network systems have not been encapsulated as combinators in existing frameworks. We provide dedicated combinator implementations, notably for batching, demuxing, muxing, and broadcasting, to encapsulate these unique data movement patterns. On top of AsyncNet, we build a prototype networking stack which is split into layers of independent protocol layers. Thanks to the specialized scheduler, embedded high-performance techniques, and dedicated combinators, the prototype networking stack, which is split into layers of independent protocol logic, outperforms the existing full- featured networking stack from Linux. Under the same la- tency constraints, AsyncNet's networking stack achieves 8.85x throughput on a UDP echo application and 1.9x throughput on a web server application.

8.2 Opening Up Kernel Bypass TCP Stacks

Michio Honda (University of Edinburgh, GB)

We have seen a surge of kernel-bypass network stacks with different design decisions for higher throughput and lower latency than the kernel stack, but how do they perform in comparison to each others in a variety of workload, given that modern stacks have to handle both bulk data transfers over multi-hundred gigabit ethernet and small RPCs that require low latency well? We found that even representative kernel-bypass stacks have never been compared for a set of basic workloads, likely because of difficulty to reproduce their implementation. This paper takes the first step towards answering that question by comparing six in-kernel or kernel bypass stacks. We show that existing stacks cannot handle those workloads at the same time or lack generality.

8.3 Low-Latency Dynamically Reprogrammable Data Planes

Georg Carle (TU München - Garching, DE)

While software updates typically require system reboots, leading to service downtimes, this talk addresses dynamically reprogrammable data planes. Our approach aims for reprogrammability while avoiding service degradation by integrating eBPF into the P4 pipeline, to combine the flexibility and dynamic adaptability of eBPF with the efficiency of P4. We investigate the approach for the P4 target T4P4S. The talk also explains the framework used to perform the experiments: our testbed in combination with our traffic generator MoonGen and our experiment framework pos (plain orchestrating service), which is part of the ESFRI SLICES research infrastructure.

9 Conclusion and Takeaways

When we first proposed this seminar, we assumed there was wide agreement on both the utility of offloading program logic to specialized hardware and the operating system, and on how modern servers should be built. The seminar itself produced a more murky picture: people are unsure about what program logic should be offloaded, to where it should be offloaded, and about what future servers should look like. Put differently, the seminar questioned our proposal's premise. We think this is a good thing: conference papers and industry specifications (which informed our proposal) cannot easily critique popular trends, and this is one of the core features of venues like Dagstuhl. Clearly we must conclude that the systems, networking and architecture community need to take a more critical look at the hardware and software assumptions on which we base our future systems.

Participants

- Emmanuel Amaro
VMware – Palo Alto, US
- Gianni Antichi
Polytechnic University of Milan, IT
- Katerina Argyraki
EPFL – Lausanne, CH
- Mario Baldi
AMD – Sunnyvale, US
- Tom Barbette
UC Louvain, BE
- Thomas Bourgeat
EPFL – Lausanne, CH
- Laurin Brandner
ETH Zürich, CH
- Georg Carle
TU München – Garching, DE
- Paul Chaignon
Isovalent – Rennes, FR
- Georgia Fragkouli
ETH Zürich, CH
- Istvan Haller
NVIDIA – Cambridge, GB
- Michio Honda
University of Edinburgh, GB
- Rishabh Iyer
EPFL – Lausanne, CH
- Kostis Kaffes
Columbia University – New York, US
- Antoine Kaufmann
MPI-SWS – Saarbrücken, DE
- Marios Kogias
Imperial College London, GB
- Jialin Li
National University of Singapore, SG
- Sebastiano Miano
Polytechnic University of Milan, IT
- Sue Moon
KAIST – Daejeon, KR
- Andrew W. Moore
University of Cambridge, GB
- Akshay Narayan
Brown University – Providence, US
- Rolf Neugebauer
Amazon – Cambridge, GB
- Aurojit Panda
New York University, US
- Boris Pismenny
EPFL – Lausanne, CH
- Salvatore Pontarelli
Sapienza University of Rome, IT
- Deepti Raghavan
Brown University – Providence, US
- Gábor Rétvári
Budapest University of Technology & Economics, HU
- Timothy Roscoe
ETH Zürich, CH
- Hugo Sadok
Carnegie Mellon University – Pittsburgh, US
- Rinku Shah
IIIT – New Dehli, IN
- Muhammad Shahbaz
Purdue University – West Lafayette, US
- Farbod Shahinfar
Polytechnic University of Milan, IT
- Mark Silberstein
Technion – Haifa, IL
- Satnam Singh
Groq – Mountain View, US
- Ryan Stutsman
University of Utah – Salt Lake City, US
- Dan Tsafrir
Technion – Haifa, IL
- Yiting Xia
MPI für Informatik – Saarbrücken, DE
- Wonsup Yoon
KAIST – Daejeon, KR
- Minlan Yu
Harvard University – Allston, US
- Davide Zoni
Polytechnic University of Milan, IT