

Programmable Network Data Planes

Edited by

Gianni Antichi¹, Theophilus Benson^{2,3}, Nate Foster³,
Fernando M. V. Ramos⁴, and Justine Sherry⁵

- 1 Queen Mary University of London, GB, g.antichi@qmul.ac.uk
- 2 Brown University – Providence, US, theophilus_benson@brown.edu
- 3 Cornell University, US, jnfoster@cs.cornell.edu
- 4 University of Lisbon, PT, fvrmos@ciencias.ulisboa.pt
- 5 Carnegie Mellon University – Pittsburgh, US, sherry@cs.cmu.edu

Abstract

Software-Defined Networking (SDN) started the “softwarization” of networking. By relocating the control plane onto a logically centralized machine, SDN gave programmers the ability to specify the behavior of the network directly in software, unleashing a major transformation both in the networking research community and in industry. However, a key limitation of the original SDN vision was the limited functionality exposed in protocols such as OpenFlow. Recent efforts to develop reconfigurable data planes and high-level network programming languages has made it possible to truly program the data plane – i.e., to change the way packets are processed on network devices. The ability to fully program the network-both control and data plane-is expected have a profound impact on the field of networking in the coming years. In this seminar we discussed the key questions and problems to be addressed in the next 10 years on the area of programmable dataplanes, and how they will potentially shape the future of networking. As an outcome we are now working on a research agenda to serve as the start of a discussion with networking researchers, practitioners, and the industry as a whole. This report is a first step towards that goal.

Seminar March 31–April 5, 2019 – <http://www.dagstuhl.de/19141>

2012 ACM Subject Classification Networks → Network architectures, Networks → Network protocols, Networks → Network components

Keywords and phrases programmable data planes, software-defined networks, programmable networks

Digital Object Identifier 10.4230/DagRep.9.3.178

1 Executive Summary

Gianni Antichi

Theophilus Benson

Nate Foster

Fernando M. V. Ramos

Justine Sherry

License  Creative Commons BY 3.0 Unported license

© Gianni Antichi, Theophilus Benson, Nate Foster, Fernando M. V. Ramos, and Justine Sherry

Traditional networks are complex and hard to manage. It is difficult to configure networks according to predefined policies, and to reconfigure them in response to dynamic changes. Traditional networks are also vertically integrated: the control and data planes are bundled



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Programmable Network Data Planes, *Dagstuhl Reports*, Vol. 9, Issue 3, pp. 178–201

Editors: Gianni Antichi, Theophilus Benson, Nate Foster, Fernando M. V. Ramos, and Justine Sherry



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

together. Around 10 years ago, the Software-Defined Networking (SDN) paradigm emerged and began to change this state of affairs. SDN breaks the vertical integration, separating the network's control logic from the underlying routers and switches (by means of a protocol such as OpenFlow) and promoting the (logical) centralization of network control. As such, it enabled the introduction of new abstractions in networking giving the ability to program the control plane of networks. Modern data center networks employ SDN-based techniques to simplify network management and operate at very large scale, and new networking services are now made possible – prominent examples are VMware's Network Virtualization Platform, Google's Andromeda and Microsoft's AccelNet.

Despite offering programmatic control to network operators, the original SDN data plane was limited to the protocols supported by OpenFlow. Over time, the OpenFlow specification evolved to support operator requirements, growing from 12 header fields in the original version to nearly 50 protocols in recent versions. The primary reason that OpenFlow is limited to specific “baked in” protocols is that the capabilities of switching chips are fixed at fabrication time. However, recent chip designs have demonstrated that it is possible to increase the flexibility of switch ASICs while still maintaining the terabit speeds required of networking hardware. In addition, as programming these chips is difficult – they expose their own low-level interface, akin to microcode programming – a domain-specific language, P4, was recently proposed to program network data planes (see p4.org). These advances are leading to a growing understanding of the inherent challenges related to data plane programming, resulting in further changes that promote future advances. For example, P4 was originally based on a simple architectural model, but has evolved to allow different switch architectures, aiming for stability of the language while increasing the flexibility to switch designers.

At the same time as programmable switches and programming languages such as P4 were being developed, a different group of researchers within the networking community has explored an alternative approach in which advanced data plane functionality is implemented on end hosts. This approach is often known as Network Function Virtualization (NFV). Platforms such as OpenVSwitch and Intel's DPDK framework make it possible to implement sophisticated packet-processing functions on end hosts rather than network switches, at line rates up to 10Gb/s and beyond. A key advantage of using CPUs is their flexibility, which makes it easy to adapt as requirements evolve. For example, it is straightforward to implement fine-grained monitoring of network flows or cryptographic operations – two pieces of functionality that are difficult to implement on standard switch ASICs.

In this context, the seminar on programmable data planes brought together leading practitioners from the areas of networking, systems, programming languages, verification, and hardware, to exchange ideas about important problems and possible solutions, and to begin the task of developing a research agenda related to programmable data planes. We have discussed several topics, including data plane architectures; programming languages, compilers and targets; use cases and applications; verification tools and formal methods; and end-system issues.

In the seminar we discussed questions including where packet-processing functionality should reside, how programmable data planes should evolve, how networks can benefit from these new elements, and how they can cope with the new challenges that arise. The focus was on the key challenges of the field and on the most fundamental problems to look at in the next 10 years, together with an aim to identify the “right” steps to take to move forward and the key problems to tackle next.

We have made some progress toward answering the following synergistic research questions during the seminar: What is the right division of labor between control and data plane? What are the right high-level language abstractions for programming networks, and what guarantees could we expect a compiler to provide reachability, security, or even properties as detailed as the correct use of cryptography? What is the trade-off between making more intelligent data plane architectures and the resulting network performances? Can we enhance current methods adopted to check network configuration errors with new solutions that automatically assure the absence of misconfiguration?

In the rest of the report we summarise the outcome of the most relevant discussions we had during the seminar.

2 Table of Contents

Executive Summary

Gianni Antichi, Theophilus Benson, Nate Foster, Fernando M. V. Ramos, and Justine Sherry 178

Seminar structure 183

Overview of talks 183

Keynote

Nick McKeown 183

Keynote

David Tennenhouse 184

Talk

Gabor Retvari 185

Talk

Dotan Levi 185

FlowBlaze demonstration

Salvatore Pontareli and Roberto Bifulco 186

Talk

Eder L. Fernandes 186

Talk

Stepahen Ibanez 187

Talk

Daehyeok Kim 187

Talk

Hugo Sadok 188

Summary of breakout sessions 188

Scalable and stateful in-network computing 188

Is P4 the RISC of Packet Processing? 189

Edge packet processing should have a different programming model from core hardware. 189

Can we bridge the gap between network measurements and application performance? 190

Data planes: security challenge or opportunity? 190

Will declarative language for networking ever succeed? Can we map a single program to multiple targets? 191

Switches should continue to be used as switches 191

Virtualization of programmable data planes won't work because current languages are too low level 192

It's time to redesign the traditional undergraduate networking course around data-plane programs! 193

eBPF, DPDK, XDP, FPGA, SmartNIC – what, when, why? 193

How can we improve application performance using programmable packet scheduling?	194
Summary of panel discussions and debates	195
Speed dating	195
Change my view debate	196
Summary of cross-pollination activities	197
Data Planes and Control Planes	
<i>Laurent Vanbever</i>	197
Programming Languages applied to Networks	
<i>Alexandra Silva</i>	198
Switch ASICs	
<i>Andy Fingerhut</i>	198
FPGAs and Smart NICs	
<i>Gordon Brebner</i>	198
Network Verification	
<i>Costin Raiciu</i>	199
Software Switches	
<i>Ben Pfaff</i>	200
Participants	201

3 Seminar structure

The organizers designed the seminar to be highly interactive. Most of the seminar was organized around sessions in which each participant was encouraged to actively contribute. For instance, on the first day, we did several rounds of “speed dating” to break the ice and quickly exchange information about technical ideas related to programmable data planes. In the same vein, “change my view” debates drew out issues for which there were a range of opinions in a constructive way. Breakout sessions were used to make progress on identifying open questions and possible directions for future research. The agenda also included several conventional talks including: two keynotes, six invited “cross-pollination” talks, as well as talks by each of the PhD students and several tool demos.

4 Overview of talks

4.1 Keynote

Nick McKeown (Stanford University, US)

License © Creative Commons BY 3.0 Unported license
© Nick McKeown

In his keynote Prof. Nick McKeown (Stanford University) shared his views on why we are where we are with programmable data planes and the P4 language. The keynote started with the question of why network programmability didn’t happen before. The reason presented was three-fold. First, there was a vested interest in the status quo. The second reason is the split between hardware and software researchers. Third, the fact that adding programmability to hardware is indeed a hard problem. So the next natural question was on why this (unstoppable) change is happening now? The “desire for programmability” in networking has its roots back in the 1990s, but it was not practical for performance and power efficiency reasons. But with respect to the data plane the assumption that held until very recently was that processing packets efficiently and with high performance required fixed-function hardware.

One of the first attempts to change the status quo happened in the late 1990s with Network Processing Units (NPUs). The focus was on parallelizing packet processing. Several research projects were started and many scientific papers were written on packet processing using multiple RISC cores. This improved the situation, but still at limited scale and performance. It seemed clear that applying the performance-enhancement techniques from computing directly into networking was not enough.

The community then started an introspection to answer the question: what type of processing is done networks? At the same time, OpenFlow was being proposed, and has been a learning experience to start to answer the question. While OpenFlow was limited, too fixed, and the abstraction was not right, it gave a good idea of the basic operation of network equipment. For packet processing, switches and routers look on headers and perform actions based on these fields: the match+action abstraction.

The community has followed observing this as a generalised abstraction, and asked itself whether there is a simple instruction set that allows us to describe on a high level language the desired packet processing to enable it to be compile it down to hardware targets? The question is complicated, as it is hard to figure out an instruction set for most domains, because parallelism needs to be taken into account. However, networks is a relatively easier

case, at least for the most common functions, due to the stateless and parallel nature of how switches process packets. Stateful functionality is a much harder problem, as it can break parallelism. There was anyway a feeling that programmability must come at a cost (more energy, slower processing, larger tables).

But turns out this is not really true, as most switching elements have no relation to programmability, so there is in fact no penalty. In practice, there is slightly lower power for the programmable part, while retaining the same performance. The die size is the same (because of the serial IO), and hence the cost is the same. Programmability represents a relatively small fraction of the chip: the wiring part is big and the size does not change, memory also uses a lot of space and is growing, and the ALUs, the programmable bit, is relatively small. As a result, while CPUs struggle to reach 100Gbps, the fastest switches can now process data at Tbps (Barefoot Tofino can switch 6.5Tbps, or 10 billion packets per second).

While we now have programmable data planes, building them is hard. The example of Barefoot Tofino, the first programmable switch, is illustrative. Tofino has 10 billion transistors, and the new version will have 21 billions. The Tofino team had around 150 people working, divided into compiler, applications, and chip design. To overall cost to design and develop a new chip is of between 150 to 250 million dollars, as the processing technology is extremely expensive and complicated. From an investor perspective this may not be very attractive because of the costs and risk.

To close, the question of inevitability was made: was programmability inevitable? The consensus was that SDN was inevitable. Big companies such as Google wanted to build cheaper networks to scale better, wanted more control, which required disaggregation, so it basically changed the scenario, and in consequence the industry. The same trend seems to be true with programmable data planes today. One of the important questions in the next 10 years is on which is the right hardware architecture for programmability.

4.2 Keynote

David L. Tennenhouse (VMware – Palo Alto, US)

License  Creative Commons BY 3.0 Unported license
© David Tennenhouse

In the second keynote, David Tennenhouse, VMware’s Chief Research Officer, has drawn a big picture of where we are in programmable networks as a result of 4 decades of introducing software into the field. A common theme was the idea of putting Computer Science into networking, as well as the role of virtualization as an enabler for innovation. An important point was the need to understand what is programmable, as this determines who can innovate and where.

David started with some historical background on the Internet, ATM and active networks. Several important concepts for programmability and virtualization were highlighted, including multiplexing (virtualizing links), the granularity of computation (per byte, per flow), virtual switching and datagrams, etc. Active networks, in particular, are very relevant as they can be considered a first instance of data plane programmability. The goal was similar: to enable innovation in networking. The concept of “network capsules”, for instance, were programs were carried in packets, and the header was in practice used as a dispatch function, with the dispatch parameters in the header. The idea was radical, in order to make room to have others make less wild (but maybe more practical) ideas.

The discussion that ensued included how the concept of active networks affected the end-to-end arguments, and what type of computation we add to the network, and where. There was a general agreement that complex computation will probably stay at the edge. IXPs were also mentioned as a good place to innovate. NFV was also discussed, an effort telcos are leading. It was argued that the problems of NFV are in many cases rooted on a lack of CS principles in addressing the existing challenges.

At closing, some of the opportunities and challenges on the field were discussed, including the need to add support for virtualization in switches, addressing the new security issues that arise in this new context, VNF integration, and the need to focus on declarative approaches and on correctness and verification mechanisms and tools.

4.3 Talk

Gabor Retvari (Budapest University of Technology & Economics, HU)

License © Creative Commons BY 3.0 Unported license
© Gabor Retvari

Gabor Retvari from Budapest University of Technology & Economics has gave a talk on “The quest for a sane definition of dataplane programming”. Gabor’s hypothesis was that a common understanding of what data plane programmability was largely missing. He presented several examples that could be perceived as such (e.g., changing BGP configurations, changing a protocol in Linux, adding a new ebpf program, etc.) and there was indeed no global agreement.

Gabor presented his personal take, that consisted in distinguishing the semantics from the behaviour. Dataplane programming would require altering the semantics (i.e., parsers/de-parsers, match-action table definitions, action types, queuing disciplines, the control flow), and using a standard interface/API exposed by the network dataplane for this purpose.

The discussion that ensued included the different roles between the data plane and the control plane, and their interactions, and the idea that to be programmable you would need to program the forwarding function of the equipment – something that was rebated with an example: what if you had a huge, but fixed, header, and the program was in the header?

4.4 Talk

Dotan Levi (Mellanox Technologies Ltd. -Yokenam, IL)

License © Creative Commons BY 3.0 Unported license
© Dotan Levi

Dotan Levi from Mellanox Technologies has given a view on the common architecture of modern programmable NIC hardware. Besides the expected Ethernet ports, buffers, and a PCIe switch, the architecture includes (R)DMA, QPC (to maintain context of the flow), schedulers, and FPGA-based accelerators for erasure coding, TLS, IPSec, etc. For the latter, there is a queue to access the FPGA, which means that from the point of view of the NIC there is only a queue. The architecture also includes a message channel to allow for disaggregation, which means the FPGA acceleration can be remote.

4.5 FlowBlaze demonstration

Salvatore Pontarelli (University of Rome “Tor Vergata”, IT) and Roberto Bifulco (NEC Laboratories Europe – Heidelberg, DE)

License © Creative Commons BY 3.0 Unported license
© Salvatore Pontarelli and Roberto Bifulco

Programmable NICs allow for better scalability to handle growing network workloads, however, providing an expressive, yet simple, abstraction to program stateful network functions in hardware remains a research challenge. Salvatore Pontarelli (Axbryd/CNIT) and Roberto Bifulco (NEC Laboratories Europe) presented a demonstration of FlowBlaze, an open abstraction for building stateful packet processing functions in hardware. The abstraction is based on Extended Finite State Machines and introduces the explicit definition of flow state, allowing FlowBlaze to leverage flow-level parallelism. FlowBlaze is expressive, supporting a wide range of complex network functions, and easy to use, hiding low-level hardware implementation issues from the programmer. They have shown their implementation of FlowBlaze on a NetFPGA SmartNIC.

4.6 Talk

Eder L. Fernandes (Queen Mary University of London, GB)

License © Creative Commons BY 3.0 Unported license
© Eder L. Fernandes

Eder Leao, Research Assistant and PhD candidate at Queen Mary University of London, presented his work “Horse: a Tool for Dynamic and Faster Experimentation of Control Planes”. The starting point is that evaluating new control plane applications or assessing the impact of changes on the network behavior, e.g., new routing protocols or topologies, requires simulation or emulation tools capable of providing results as close as possible to those from a real-world experiment. Large traffic loads and dynamic control-data plane interactions constitute significant challenges to these tools. In addition, the need for ever-increasing computational resources severely limits researchers’ ability to test their ideas at scale. To address these challenges Eder proposed a tool for faster evaluation of both SDN and legacy networks. His approach emulates the network’s control plane and simulates the data plane, to bring realism to control plane testing while being also capable of experimenting for increasing topology sizes and loads. He presented the design and implementation of a proof of concept, and some preliminary results. The discussion centered around some of the challenges of the approach, including control plane convergence, issues with the fluid traffic models used, and the time to setup the control plane.

4.7 Talk

Stephen Ibanez (Stanford University, US)

License © Creative Commons BY 3.0 Unported license
© Stephane Ibanez

The rise of P4 programmable devices has sparked an interest in developing new applications for packet processing data-planes. Unfortunately, the application developers have only met with limited success. The bane of designing data-plane applications is that they must satisfy the strict constraints of the underlying hardware, which makes it difficult to implement complex stateful processing logic. These challenges are exacerbated by the fact that modern programmable data-plane architectures support a very small collection of events. Typically, packet arrivals, packet departures, and recirculation events. In his talk “Event-Driven Packet Processing” Stephen Ibanez, PhD student at Stanford University, shared his observation that network applications are inherently event-driven and as such, programmable data-plane hardware should also be event-driven. By identifying a set of useful data-plane events and outlining a new hardware architecture to support them, he demonstrated how we can achieve an unprecedented level of programmability without sacrificing performance. The discussion centered around some of the potential consistency issues of the solution.

4.8 Talk

Daehyeok Kim (Carnegie Mellon University – Pittsburgh, US)

License © Creative Commons BY 3.0 Unported license
© Daehyeok Kim

Programmable switches have been touted an attractive vantage point to serve various network functions (NFs) such as network address translators, load balancers, and virtual switches because of their in-network location and high packet processing rates. However, the limited memory capacity on programmable switches has been a major stumbling block that has stymied their adoption for supporting many memory-intensive NFs (e.g., datacenter scale NATs and load balancers that maintain millions of flow table entries). While it is theoretically possible for switch ASIC vendors to extend the memory capacity (e.g., adding more SRAM or adding off-chip DRAM), these solutions are not cost-efficient and are fundamentally limited in flexibility and scalability.

In his talk “Generic External Memory for Switch Data Planes”, Daehyeok Kim, PhD student at CMU, presented an alternative approach in which NFs implemented on a programmable switch can make use of DRAM on servers connected to the network. The design is driven by the observation that in data centers, DRAM and network resources are underutilized. This gives an opportunity to leverage those unused resources to extend switches’ memory capacity with low cost. While this low cost solution is appealing, there are several technical challenges before we can realize this in practice. These include performance, load balancing, and fault-tolerance. In the talk, he described the key challenges, how they addressed them, also introduced his prototype implementation and preliminary evaluation results to demonstrate the practicality of the solution.

4.9 Talk

Hugo Sadok (Federal University of Rio de Janeiro, BR)

License  Creative Commons BY 3.0 Unported license
© Hugo Sadok

The standard approach adopted by software middleboxes to use multiple cores has long been to direct packets to cores at flow granularity. This, however, has significant shortcomings. First, it is inefficient, since it cannot use all cores when there is a small number of concurrent flows—which happens frequently. Second, asymmetry in flow distribution causes unfairness even with a larger number of flows. Yet, the current trend of higher-speed links and core-richer CPUs only aggravates these problems. In his talk “A Case for Spraying Packets in Software Middleboxes”, Hugo Sadok, PhD student at CMU, proposes a natural alternative: that middleboxes should direct packets to cores at a finer granularity. His system, *Sprayer*, solves the fundamental problems of per-flow solutions and addresses the new challenges of handling shared flow state that come with packet spraying. *Sprayer* builds on the observation that most middleboxes only update flow state when connections start or finish; ensuring that all control packets from the same TCP connection are processed in the same core. In the talk he has shown that, when compared to the per-flow alternative, *Sprayer* significantly improves fairness and seamlessly uses the entire capacity, even when there is a single flow. The discussion centered around potential reordering issues from the cumulative effect of chaining network functions.

5 Summary of breakout sessions

5.1 Scalable and stateful in-network computing

Support for stateful packet-processing is an important consideration that differentiates various programmable data planes. Early efforts focused on turning fixed-function switches to programmable switches, maintained the classic separation of control plane and data plane, meaning that table updates can only be performed using the control plane. Furthermore, it assumed that operations within the data plane are stateless or mostly stateless (e.g., support for MAC learning, or packet and byte counters might be provided). However, the development of the PISA architecture, which provides support for general-purpose registers, as well as the emergence of in-network computing generalizes the simple SDN model. However, supporting network functions that require per-flow state such as NAT and load-balancers remains challenging.

Going beyond PISA, there have been several recent attempts to enable stateful operations in the data plane. One such example is *FlowBlaze* [NSDI’19], which uses state machines to enable stateful operations. *P4->NetFPGA* [FPGA’19] enables stateful operations using externs, and supports atomic operations for maintaining and changing state.

Overall, the group concluded that support for stateful operations within the data plane is essential. To achieve this will require introducing new primitives to P4 and related languages that abstracts away the details of the hardware while still supporting mutable state. A key challenge will be adding stateful operations without affecting performance: switches must still support full line rate with only minimal changes to existing hardware designs.

5.2 Is P4 the RISC of Packet Processing?

The discussion started with the evolution of programmable networks, from pre-SDN to programmable data planes and the evolution of the P4 language. In an analogy with the evolution of human societies, before SDN was the hunter-gatherer time period, with hardware architects in charge of networking. The SDN period was the period after the “discovery of fire”, by giving more control to the users, but still with the data plane dictated by the manufacturer: fixed parsers and fixed match-action. The P4_14 period corresponds to the “domestication period, separating planting from harvesting”, introducing programmable parsers, programmable match-action, and primitive actions. Finally, the P4_16 period was the period of a “sophisticated civilization (e.g. roman empire)”, with an enhanced version of the language, and the introduction of the Portable Switch Architecture (PSA) and API generation framework (control plane API, such as P4Runtime).

At the moment, P4 seems to be an ISA-like interface between higher-level applications (L3 router, telemetry, stateful applications) and the underlying target architectures (software switches, FPGA, Tofino, SmartNICs, etc.). The higher level applications are compiled down to the ISA (P4 + PSA) – we could call it RISN (Reduced Instruction Set Networking) – and the target architectures implement the interface.

5.3 Edge packet processing should have a different programming model from core hardware.

While everyone agreed on what constituted a network “core”, the edge became far more contentious. The focus shifted to discussing different granularities of where programmability is inserted at the edge. The group summarize a list of different properties of edge processing tasks, for example:

- those processing aggregates of traffic
- those processing a few flows
- those processing hundreds or thousands of gigabits
- those processing a few megabits
- those that hold entire movies
- those that only hold a few flow table entries
- those that need no state to process packets
- those that need only a fixed amount of per-flow state
- those which reconstruct the entire data in a connection and might need an arbitrary amount of state
- those that use shared state
- those that encrypt packet, or decrypt them, or compress them, or do things one cannot even imagine

Hence the conclusion was that the programming model was more tied to the capabilities required rather than core vs edge. The conclusion hence led to a new question: “What is an appropriate taxonomy of processing capabilities that we might use to derive programming models?”

5.4 Can we bridge the gap between network measurements and application performance?

The group of people participating this breakout session recognises that there is a mismatch between operators' and users' perspective. Such a mismatch results in a non-trivial translation from the data retrieved with network monitoring practices and application needs. On the one side, the collection of monitoring information has been so far a pretty well explored area by the community. Many different approaches have been considered, from flow-level analysis to per-packet measurements using probabilistic data structures, i.e., sketches, or In-band Network Telemetry (INT). On the other side, there is still a lack of consensus on high-level abstractions. The group agreed that although this is somewhat an old problem, network programmability can help in the process of aggregating fine-grained measurements into high-level application-aware metrics. Specifically, bridging the gap between network measurements and application performance can be achieved only through rethinking application development process. Indeed, just capturing and then analysing all the traffic in the network, which is a very challenging problem itself, might not be enough to have a comprehensive understanding of the network/applications ecosystem. This is because the translation to high-level metrics, e.g., service level objectives (SLO), remains still unclear. To bridge this gap, network packets need to be easily associated to specific applications. This can be done by tagging the packets. In this way, applications can "dialogue" with the network and inform which packets are important or which metrics affect mostly the correct behavior, i.e., latency, jitter. As a consequence, there is a need to rethink application development process: developers need to provide information about network metrics they are interested in and subsequently embed those notions in form of tags in the packets. Then, leveraging data plane programmability network operators can use the tags to translate the fine grain measurements taken in the network with the high level objective of an application. Such an approach will also help during the debugging process when an application performance degradation is experienced by quickly locating if the problem relates to the network or the software application itself.

5.5 Data planes: security challenge or opportunity?

In this breakout session we started by discussing the security challenges of programmable data planes. First, software is more bug prone than hardware. Programmers tend to take it more light heartedly, and software is to be written by the end user, that is usually less experienced (and careful) than a vendor. The functionalities will also tend to be more complex, potentially creating new problems. Second, attackers can now change the behavior/semantics of the device. Similarly to SDN, this creates new attack vectors that have to be addressed in a different way from traditional measures. Finally, it becomes important to make sure that countermeasures are applied (e.g., assertions to be added to the program, verification be performed, etc.).

Then, we discussed how it could improve security. First, it allows the development of new verification mechanisms and tools. This requires different objectives than functional verification, and proper security models. In addition, it needs to be updated as we add security functions, potentially at the network level (vs just at the device level). Second, it enables containing effects of attacks ("sandboxing"), for examples by using (micro)segmentation of the network, traffic, code, and of the execution environment. Third, we can now include assertions to be added by security experts (vs programmer), that are not program specific and can possibly be common to multiple programs (e.g., aspect-oriented programming). The expressiveness of the language may need to be restricted for security reasons.

Finally, we addressed opportunities, including rapid remediation or minimizing the number/amount of functions on the device. The fact that assertions and verification are enabled give opportunities for hardening networks. In addition, the end user has now visibility on the program, so it can fix security breaches directly. Open source can also be an opportunity (and a challenge). There are also new opportunities for the network to protect the rest (e.g., in the IoT case), as programmability allows implementation of more sophisticated countermeasures. The biggest challenge is to take advantage of the opportunities.

5.6 Will declarative language for networking ever succeed? Can we map a single program to multiple targets?

Writing code for network devices require a deep understanding of the capabilities supported by the target architecture. The level of functionalities exposed to the programmer, such as supported match types and hash functions, or Read-Modify-Write operations (RMW) that can be performed vary greatly across distinct chips. Considering the fact that the desire of developers for portability not always align with vendor's need for differentiation, we raise some relevant points that should be taken into account on the quest for higher level languages to program data planes.

We first look into stateless packet processing. Because the trade-offs of programmability for stateless header processing are almost nonexistent, it should be feasible and easier to have a well defined minimum set of features that must be supported across different targets (a Portable Switch Architecture). Such definition could enable the development of portable programs without giving performance away.

Stateful packet processing is a more complex problem. The need to store and keep track of network information is different across different applications, what makes it harder to define a standard set of instructions. Such complexity raises questions:

- Is there a minimal set of instructions that could possibly become a standard Instruction Set Architecture (ISA)?
- It is acceptable to exchange performance for generality? Is there an acceptable threshold for most applications?

For the first question, if we think about traditional stateful network applications such as firewalls and NAT, there are some well defined actions needed to implement the applications. However, new applications for in network computing could bring a whole new set of specific instructions. As for the second question, a potential approach would have compilers able to estimate and expose the potential loss of performance and let the customers weigh on the trade-offs.

5.7 Switches should continue to be used as switches

The discussion within this breakout group on “Switches should continue to be used as switches, i.e., as network devices that receive packets, parse them, and then forward them with little or no modification” evolved around the general role of switches in today's networks: whether switches should simply forward packets or also perform more complex operations. Several recent works have shown how programmable switches can be used to improve application performance in two possible ways: i) improving the communication channel through which

applications communicate (i.e., the network/transport layer) or offloading application logic into the network. Through the first approach, network operators can program data plane pipelines to support the delivery of a packet from a source node to its designated destination while guaranteeing some level of performance. For instance, operators rely on load balancers to make efficient utilization of network bandwidth resources, use network telemetry to verify the status of the network communication, implement fast reroute to quickly detour packets upon network failure, and use packet schedulers and congestion control support to implement different level of application performance in the network. Programmable data-planes are clearly a game changer for this type of network applications. Through the second approach, network operators can program part of the application logic within the data plane packet processing pipeline. Some examples of this paradigm include range from key-value storage systems (e.g., NetCache [1]) to coordination services (e.g., NetChain [2]) and beyond. This group believes that in-network computing holds great promise for programmable data planes but it yet has to prove its advantages with convincing use cases, which have been so far limited and debatable. Today's main barriers to move application logic into the network are represented by the limited memory support at data plane speed (e.g., small, no transactional) and lack of transport reliable support, which makes it harder to implement distributed mechanisms. Whether switches will be used in the future to offload application logic clearly depends on a variety of factors including costs, future application requirements, and switches data plane capabilities. We however expect to see extensive research efforts coming from the networking community in the study of in-network computing in the coming years.

References

- 1 X. Jin et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching". In SOSP'17.
- 2 X. Jin et al. "NetChain: Scale-Free Sub-RTT Coordination". In NSDI 2019.
- 3 J. McCauley et al. "Thoughts on Load Distribution and the Role of Programmable Switches". In CCR 2019 (Editorial note)

5.8 Virtualization of programmable data planes won't work because current languages are too low level

The discussion of this breakout group started by noticing that there is no such a thing as "too low level language" for virtualisation, i.e., assembly. Then, the discussion has evolved around the meaning of virtualisation itself. In this context, the group has focussed its attention on the problem of sharing switch's resources among different programs. Although enabling composition of different P4 programs has already been tackled by existing literature, i.e., P4Visor [1], the group acknowledged that it is not clear yet how to support performance isolation. One option can be to leverage the multiple pipelines available in current programmable data planes hardware and switch between them by recirculating the packets. This solution has been considered not practical though. Indeed, it was noticed that different pipelines are generally attached to different physical ports, making the switching of pipelines possible only if cables closed in loopback are being adopted, thus wasting precious physical port resources. After some brainstorming around potential solutions and their drawbacks, the group has agreed that the P4 language should not represent a barrier for virtualisation. In contrast, the specifics of a programmable hardware could, although those can reflect back to the language itself. Indeed, a meaningful question seems to be whether the underlying architecture has features that hinder virtualization, e.g., one might want to have multiple

queues that are assigned to different slices and each slice has its own scheduler under the control of one single program. This is a call for a virtualisation-aware hardware architecture that builds upon the PIFO results [2].

References

- 1 P. Zheng et al. “P4Visor: lightweight virtualization and composition primitives for building and testing modular programs”. In ACM CoNEXT 2018
- 2 A. Sivaraman et al. “Programmable Packet Scheduling at Line Rate”. In ACM SIGCOMM 2016

5.9 It’s time to redesign the traditional undergraduate networking course around data-plane programs!

The breakout session started with the question that we need to be clear about what we mean by networking, as there are many aspects (e.g., physical layer, coding theory, etc.). The focus was on Computer Science courses, and they are mainly about the way the Internet works. The reason is probably the Internet ossification, so for an introductory class you want to teach students how sockets work, explain Ethernet, ARP, DHCP, DNS, etc. This ends up as becoming in the awkward middle between theory-style classes and system-style classes.

Given the layered structure of the Internet, several approaches have been tried. The most common today are Kurose and Ross’s top-down approach and Peterson and Davie’s bottom-up, but others (e.g., Scott Shenker) have a different approach: starting from the middle, with routing and reliable transport. And then spreading to the edge. Trying to frame the lecture as a solution to a problem that we can define clearly was seen as an interesting approach. The lecture starts by presenting the students a strawman solution, and then refine it with their help.

With respect to the laboratorial sessions, several practical assignments and group projects were discussed. Some (Jon Crowcroft a while ago) involved writing a new transport protocol and interoperate. Others (Laurent Vanbever at ETH) include each group of students managing their own AS, requiring them to configure their own networks and establishing connectivity with others groups/ASes, using Docker, OVS, Quagga, etc. In Stanford and Cambridge there is also one course on how to build a router that is of interest (currently using P4 which facilitates development). Congestion Control Shootout + ETH Communication Networks seem like three ingredients that get at what I’m looking for. Keith Winstein at Stanford also has a TCP congestion control contest.

P4 can be a game-changer in this respect, and the P4 education group has been making an effort to publicly share curated material in their github page.

5.10 eBPF, DPDK, XDP, FPGA, SmartNIC – what, when, why?

These are very diverse technologies that cannot be compared to each other directly: DPDK is a framework, eBPF is a virtual machine, XDP is a hook, FPGA is hardware, and a SmartNIC can be many different things. However, they can potentially be used to implement similar applications (or parts of them), so we summarize the advantages and disadvantages of each.

DPDK has the advantages of being an industry standard for NFV, and industry backing. Enables good software-level performance with many optimizations such as pre-fetching, memory, caching, vectorization, SIMD for parallel workloads, etc. As disadvantages, it

requires domain specific knowledge, dedicating ports, and rewriting the network stack in userspace, and/or looping back to the kernel.

eBPF's advantages include being integrated with the Linux kernel, can be changed without changing the kernel, although running in the kernel, and good performance if using the right hook. However, it has somewhat limited functionality, it's difficult to implement complex functions (Cilium uses many helper functions in the kernel, for instance), and does not expose low-level code to acceleration, making it hard to take advantage of vector capabilities like SIMD instructions; especially since hooks are called per packet.

XDP is one of the hooks for eBPF, and the recent introduction of AF_XDP allows eBPF to combine user space and kernel packet processing (essentially providing an early branching point between the two). As advantages, it has near DPDK performance without dedicating a NIC like DPDK, and is good if you want to use the Linux kernel to multiplex against different network stacks. As disadvantages, it still needs to do everything to the packet in userspace (like DPDK).

FPGA and SmartNICs are very diverse technologies, with no clear baseline and lots of implementation specific features. The best aspects are that it is not needed to burn a core for performance, and helps in meeting space constraints (i.e., no space to deploy extra servers in some edge deployments). However, it is unclear how to program them (no real standard), platforms are very different, and they often do not have guaranteed performance.

5.11 How can we improve application performance using programmable packet scheduling?

The breakout session started by asking problems does packet scheduling solves? First, bandwidth guarantees, isolating well behaved flows from ill-behaved flows. Then, minimizing delay (latency), FCT (with and without flow size info), offering delay bounds (both local and end-to-end) and delay jitter bounds. It helps minimizing the number of missed deadlines (or deadline violation probability bound), offers bounded loss rate (or buffer overflow probability bound), minimize slowdowns (a.k.a. bit transmission delay) – $\text{delay} / \text{flow_size}$, and allows to simultaneously achieve multiple performance objectives.

Application need to provide some information to the network to help it make scheduling decisions, such as indicating co-flows so that the network can implement policies that minimize co-flow completion times, which packets are more favorable to be dropped, provide back pressure into the network so the network can use that info to make scheduling decisions (e.g., deadline hints). Several efforts exist to achieve this goal, including the TAPS IETF Working group that is discussing ways to change the socket interface so that applications can indicate performance requirements/desires to the network.

Some applications might be improved by enabling custom packet scheduling policies, including network performance isolation (the network could prioritize packets from flows with no downstream congestion in order to avoid sending packets that will be dropped downstream, thus wasting capacity reducing isolation), and Time Sensitive Networking (TSN), that often requires precise synchronization and packet scheduling amongst switches.

6 Summary of panel discussions and debates

To spark active conversation and open-ended discussion, we started the seminar with several semi-structured activities between small groups.

6.1 Speed dating

In the first activity, “Speed Dating”, individual researchers led small group discussions. Each leader proposed a (semi-controversial) thesis statement and gave a short 5 minute “pitch” in favor of their thesis to a small group of 5-7 people. The small group then asked questions or proposed alternative theses and argued against the proposal. After about 10-12 minutes, conversation stopped and each small group member moved to a different leader. The leaders then repeated their “pitch” and led another discussion. This process of 10-12 minute small group discussions repeated until every attendee had visited every leader’s presentation. Some of the presenters and thesis statements were as follows:

Minlan Yu (Harvard University): “Measurement system design should be led by applications not network devices”. Today, many measurement systems (e.g., the latest INT design) are all led by capabilities of network devices, followed by operators who then think about what queries they can ask using the new measurement capabilities. I argue that we should design measurement systems focusing on how to better serve applications directly without worrying about device capabilities. We have entered a stage where we have devices with enough programmability and researchers/engineers who are smart enough to implement what we need if we have a good abstraction that meet application needs.

Justin Pettit (vmWare): “AF_XDP is the right way to do dataplane processing in Linux” Dr. Pettit aimed to focus the conversation on a sequence of questions related to the AF_XDP extension for dataplane processing: What are the security implications, specifically in regards to side-channel attacks? At what point (specifically speed) does it make sense to start offloading some or all to hardware? And if we want to move it to hardware, what does that look like? FPGA? SmartNIC with small general purpose CPUs? NIC with flow offloading?

Timothy Griffin (Cambridge University): “Dynamic Routing Protocols Work!” Dr. Griffin provided the following summary: This was an intentionally provocative assertion. It generated a conversation that continued throughout the week with many people. Gradually, it morphed into this question: “In the future when programmable switches are ubiquitous, what changes will we see in the IETF?” Presumably it will make some kinds of interoperable protocols easier to define, develop, and test. We may see many more network operators and end-users proposing protocols. This may be especially true in the areas of transport and security. However, in the control plane things may be different. What incentives are there now for network operators to worry about interoperability? Inter-domain, OK, sure. But intra-domain? Not clear. Perhaps end-users will force some minimal interoperability so that services can span multiple providers (for example, VPN standards such as VXLAN and other virtual network services). My assertion is that this kind of activity represents a “paradigm for programming the dataplane” – new functionality is introduced in the dataplane (say some kind of tunnel), and distributed protocols are developed and defined to populate the in-switch tables needed to implement the functionality. I don’t think this paradigm is going away any time soon.

Noa Zilberman (Cambridge University): “In-network computing is the way to scale computing” Dr. Zilberman provides the following summary: In network computing is the execution of traditionally host-based applications within the network, e.g., within a switch

or a NIC, as the traffic goes through the network. Network devices process billions of packets a second, and several works have demonstrated their use to achieve billions-of-ops a second throughput (KVS, consensus, data aggregation, network services etc.). While CPUs have improved x27 in performance since they met the end of Moore’s (and Dennard’s) law, network devices have scaled x200, and (so far) continue and scale [2015 numbers]. In-network computing provides a trajectory for continuing and scaling computing, providing the much needed improvement in performance (throughput, latency), alongside better power consumption (10M’s of ops/Watt in a switch, compared with 10K’s of ops/Watt on a server) they are also “low cost” – you already have the device within your network.

6.2 Change my view debate

In the second activity, we conducted a debate style conversation with two individuals leading the conversation and presenting arguments for and against a specific topic.

The first topic was: “Deep Packet Inspection is obsolete. The rise of end-to-end encryption renders DPI useless.”

Aurojit Panda was for, with three arguments. First, that it was unsafe, as there is a lot of TLS interception happening, there is the problem of old cipher suites, and that generally adding boxes is making us worse. Second, it is infeasible, due to the stateful requirement that does not allow it to scale. Finally, it is unnecessary, as we need the DPI to have policy, and content analysis is hard – we should just use meta-data.

Jon Crowcroft was against, as we need tattered ends for latency, energy, and privacy. We need at least a bit of DPI, as serving content from the edge is important.

Others have argued TLS is everywhere, so its feasibility is killing DPI, and software developers are winning because you have it at the edge; there should be active delegation of trust, so things should not be done surreptitiously; and there is a tension between the stakeholders. Would do we trust? Who checks our network? Should anyone take care of security on my behalf?

The second topic was “P4 will stand the test of time. (20 years from now we will all be writing P4v39)”

Robert Soule is for, arguing PISA to be the natural design. Also, P4 is simple, follows well the match-action abstraction, and these fundamental abstractions will stay with us. Sujata Banerjee was against, as she thinks we should be programming at a higher level of abstraction. Also, maybe we won’t be programming data planes in 20 years, as they may have all been written.

Robert rebuttal mentioned programming in P4 to be painful and that we indeed need a higher level abstraction, but we will develop that with time and P4 may be the assembly code and other languages will compile to P4

Others argued that we may not need P4 in 10 years as the network will be so powerful that we will have functions at the edge. Others have said it’s inevitable because it restricts the right way, its extensible, and has clear semantics.

The third topic was “FPGAs and SmartNICs will or should take over the role that x86 plays in software packet processing, e.g., software switching and network functions”

Dotan Levi is for has FPGAs give diversity, and ASICs need to be generic, so there are things the FPGA will not do. Ben Pfaff is against as FPGA was always the “tech of the future”, but has important limitations, such as many customers wanting L4 stateful, with FPGAs helping a little but not enough. Also, above L4 you need to match in URLs, etc., so NetFPGAs also does not help here. Finally, FPGA programming is not portable, and it’s challenging to hire skilled people.

Some have argued FPGA to be good for parallelism, and other mention the problem of not being power efficient. Other mentioned that CPU memory hierarchies can now be written and not reverse engineered which is positive. Finally, new FPGA-based products are not only FPGA anymore (e.g., Xilinx ACAP). There is programmable logic; there is a processor; blocks for ML, etc.

7 Summary of cross-pollination activities

7.1 Data Planes and Control Planes

Laurent Vanbever (ETH Zürich, CH)

License © Creative Commons BY 3.0 Unported license
© Laurent Vanbever

Should network control planes be centralized or distributed? Should they be implemented in software or in hardware? In this talk, Prof. Vanbever presented his recent journey exploring these design dimensions. He first described Fibbing [1] and Netcomplete [2], two frameworks enabling to control distributed network control planes in a centralized manner by synthesizing: (i) the routing announcements; or (ii) the router configurations, respectively. He then described Blink [3] and hardware-accelerated control planes [4], two recent works that show the benefits of offloading pieces of the control plane logic (e.g., detecting failures) to the data plane. Specifically, Fibbing introduces fake nodes and links into an underlying linkstate routing protocol, so that routers compute their own forwarding tables based on the augmented topology. Fibbing is expressive, and readily supports flexible load balancing, traffic engineering, and backup routes. Netcomplete is instead a system that assists operators in modifying existing network-wide configurations to comply with new routing policies. NetComplete takes as input configurations with “holes” that identify the parameters to be completed and “autocompletes” these with concrete values. Finally, the talk showed that programmable data planes are powerful enough to run key control plane tasks including: failure detection and notification, connectivity retrieval, and even policy-based routing protocols. As an example, Blink is a data-driven system exploiting TCP-induced signals to detect failures. The key intuition behind Blink is that a TCP flow exhibits a predictable behavior upon disruption: retransmitting the same packet over and over, at epochs exponentially spaced in time. When compounded over multiple flows, this behavior creates a strong and characteristic failure signal. Blink efficiently analyzes TCP flows, at line rate, to: (i) select flows to track; (ii) reliably and quickly detect major traffic disruptions; and (iii) recover data-plane connectivity, via next-hops compatible with the operator’s policies.

References

- 1 S. Vissicchio et al., “Central Control Over Distributed Routing”, ACM SIGCOMM 2015
- 2 A. El Hassany et al., “Netcomplete: practical network-wide configuration synthesis with autocompletion”, USENIX NSDI 2018
- 3 T. Holterbach et al., “Blink: fast connectivity recovery entirely in the data plane”, USENIX NSDI 2019
- 4 E. Costa Molero et al., “Hardware-Accelerated Network Control Planes”, ACM HotNets 2018

7.2 Programming Languages applied to Networks

Alexandra Silva (University College London, GB)

License  Creative Commons BY 3.0 Unported license
© Alexandra Silva

Dr. Silva provided an overview of programming languages applied to networks to an audience of 12-16 attendees. Her talk combined principles (e.g., desirable properties for a network programming languages) as well as the presentation of a specific network programming language: NetKAT. She introduced the concepts of semantics (both denotational and operational), expressiveness, verification, and extensions. She then described KAT, “Kleene Algebra with Tests”, and showed how to apply KAT to NetKAT which allows network operators to prove properties such as reachability. The audience discussed how KAT/NetKAT, based in regular expressions, was a straightforward theoretical framework both for systems audiences and for theorists due to the ability to describe the language in terms of finite automata. Dr. Silva then briefly described ProbNetKat and ConcurrentProbNetKet to describe probabilistic events like congestion. The discussion ran over time and led into smaller group Q&A.

7.3 Switch ASICs

Andy Fingerhut (CISCO Systems – San Jose, US)

License  Creative Commons BY 3.0 Unported license
© Andy Fingerhut

Most packet processing devices today are general purpose CPUs, but most of the bits/second pass through specialized devices like NPUs and configurable switch ASICs (a term used to call out that while they have many configuration options, they typically are not programmable in the way a Barefoot Tofino or NPU are). In this cross-pollination session the discussion ensued around these questions: What are some of the reasons for the wide differences in cost per port for different device categories? What can the configurable switch ASICs do? Why are some of them so complex, and why is the control plane software developed by vendors like Cisco so expensive to develop and maintain?

7.4 FPGAs and Smart NICs

Gordon Brebner (Xilinx – San Jose, US)

License  Creative Commons BY 3.0 Unported license
© Gordon Brebner

In his cross-pollination session, Gordon Brebner from Xilinx Labs has discussed FPGAs and Smart NICs. After an introduction to FPGAs, Gordon has discussed the advantage of the Adaptive Compute Acceleration Platform in modern FPGAs, that drastically reduces the programmability cost. It was made clear that the FPGA is not just the logic gates, including a CPU, Network on chip, an AI Engine array, etc. A discussion ensued on the difficulties of programming an FPGA, namely the need to think like a hardware designer, describing the design in Verilog and VHDL, and having to worry with circuit size and operating at a certain

clock rate. The good news is that today the hardware design experience is less needed, given the emergence of software-side languages and libraries, including domain-specific (e.g., P4) or general-purpose (e.g., C).

Many layers (of hardware, software, and expertise) are needed to build a network program: starting from the chip designer and FPGA hardware programmer, moving to the P4 pipeline architecture and P4 data plane, and ending on the P4 run time programmer and application programmer. This effort is now being used with tools such as the P4 for FPGA compiler (joint work from Xilinx Labs and Stanford University).

Next, Gordon presented some ideas on NICs, from the basic NICs (e.g., from Intel) to smart NICS (both FPGA- and NPU-based), contrasting them with switches. Interestingly, their architectures are pretty similar. The evolution of Xilinx NICs was only presented, starting from NICs with standard network functions, and their evolution to include custom functionality, including acceleration elements (e.g., for transport functions), increasing architecture disaggregation, and moving from packet processing to stream processing.

Gordon closed the session by presenting some of the issues being discussed around the P4 Portable NIC Architecture. These include, first, a discussion on the standard components of the ingress and egress pipeline, whether more variability was needed when compared to the Programmable Switch Architecture (PSA), and the isolation and virtualization mechanisms required. Second, the question of how the host CPU interface can be modelled, how to differentiate between the data plane CPU and the control plane CPU, and the impact on P4Runtime. Third, thinking beyond packet forwarding. For instance, is protocol (e.g., TCP) termination covered? Should we perform payload processing as well as forwarding? Finally, the aim is for a modular specification that has as much in common with PSA as possible.

7.5 Network Verification

Costin Raiciu (University Politehnica of Bucharest, RO)

License © Creative Commons BY 3.0 Unported license
© Costin Raiciu

Costin Raicu presented an in-depth introduction to verification of stateful data planes. In the first part of the talk, he discussed two verification approaches: one based on symbolic execution and one based on weakest preconditions. Both approaches rely on a first-order solver (e.g., Z3) to check the satisfiability of logical formulas. But they differ in how those formulas are constructed. Symbolic execution traverses the program in the forward direction, while weakest preconditions propagates logical conditions backwards through the program. These approaches perform differently depending on the characteristics of the program.

In many programs, it is important to model assumptions about the control plane to allow verification of the data plane to succeed. Existing tools usually require the programmer to add manual annotations to the program that capture these assumptions. In the second part of the talk, Costin discussed a new approach that can automatically synthesize the necessary preconditions required for executing a given table. In practice, this technique enables verifying properties such as header validity without any manual intervention on the part of the programmer.

7.6 Software Switches

Ben Pfaff (VMware – Palo Alto, US)

License  Creative Commons BY 3.0 Unported license
© Ben Pfaff

Open vSwitch lacks support for user-configurable protocols: users cannot easily add support for new protocols or fields within existing protocols. The PISCES project [1] from a few years ago showed one way to do this with P4, but PISCES could not be merged into mainstream OVS because it broke backward compatibility, which would not be acceptable to many OVS users. Additionally, it did not support the kernel datapath. In practice, the biggest obstacle to adding P4 or other user-configurable protocol support is the OVS datapath interface. This interface is a frozen ABI because it is implemented in the Linux kernel; it can be extended but not changed. It has a specific idea of what protocols exist, which can similarly be extended but not changed. Extensions can happen only very slowly: they must first be pushed into upstream Linux, which releases a few times a year, and then percolate slowly out to Linux distributions and then to users over a period of years. This is not practical for user-configurable nonstandard protocols. What's the way forward then? One way would be to use an eBPF program instead of a Linux kernel module, since eBPF offers more opportunity for local customization. But eBPF is currently too restrictive and has a performance penalty. Over time, these disadvantages might decline. Another approach would be to use AF_XDP for fast userspace access to packets. With that approach, OVS could drop the datapath interface entirely (as long as some similar approach could be worked out for Hyper-V), which would give OVS much more opportunity to evolve in new and exciting ways.

References

- 1 Shahbaz et al., “PISCES: A Programmable, Protocol-Independent Software Switch”, ACM SIGCOMM 2016

Participants

- Gianni Antichi
Queen Mary University of
London, GB
- Mario Baldi
Polytechnic University of
Torino, IT
- Sujata Banerjee
VMware – Palo Alto, US
- Theophilus Benson
Brown University –
Providence, US
- Roberto Bifulco
NEC Laboratories Europe –
Heidelberg, DE
- Gordon Brebner
Xilinx – San Jose, US
- Marco Chiesa
KTH Royal Institute of
Technology – Stockholm, SE
- Paolo Costa
Microsoft Research –
Cambridge, GB
- Jon Crowcroft
University of Cambridge, GB
- Lars Eggert
NetApp Finland Oy, FI
- Anja Feldmann
MPI für Informatik –
Saarbrücken, DE
- Andy Fingerhut
CISCO Systems – San Jose, US
- Nate Foster
Cornell University, US
- Soudeh Ghorbani
Johns Hopkins University –
Baltimore, US
- Timothy G. Griffin
University of Cambridge, GB
- Stephen Ibanez
Stanford University, US
- Changhoon Kim
Barefoot Networks –
Palo Alto, US
- Daehyeok Kim
Carnegie Mellon University –
Pittsburgh, US
- Eder L. Fernandes
Queen Mary University of
London, GB
- Alberto Lerner
University of Fribourg, CH
- Dotan Levi
Mellanox Technologies Ltd.
-Yoknam, IL
- Nick McKeown
Stanford University, US
- Aurojit Panda
New York University, US
- Justin Pettit
VMware – Palo Alto, US
- Ben Pfaff
VMware – Palo Alto, US
- Salvatore Pontarelli
University of Rome “Tor
Vergata”, IT
- Costin Raiciu
University Politehnica of
Bucharest, RO
- Fernando M. V. Ramos
University of Lisbon, PT
- Gabor Retvari
Budapest University of
Technology & Economics, HU
- Hugo Sadok
Federal University of Rio de
Janeiro, BR
- Justine Sherry
Carnegie Mellon University –
Pittsburgh, US
- Salvatore Signorello
University of Lisbon, PT
- Alexandra Silva
University College London, GB
- Robert Soulé
University of Lugano, CH
- Alex Sprintson
Texas A&M University –
College Station, US
- David L. Tennenhouse
VMware – Palo Alto, US
- Laurent Vanbever
ETH Zürich, CH
- Stefano Vissicchio
University College London, GB
- David Walker
Princeton University, US
- Hakim Weatherspoon
Cornell University, US
- Minlan Yu
Harvard University –
Cambridge, US
- Noa Zilberman
University of Cambridge, GB