In-Network Compute: Considered Armed and Dangerous

Theophilus A. Benson
Brown University

Abstract

Programmable data planes promise unprecedented flexibility and innovation. But enormous management issues arise when these programmable data-planes, and the in-network compute functionality they enable, are deployed within production networks. In this paper, we present an overview of these management challenges, then explore the limitations of existing management techniques. Finally, we propose a system, Harmony, that encapsulates new abstractions and primitives to address these problems.

CCS Concepts • **Networks → Programmable networks**; **Network management**.

Keywords in-network computing; programmable network devices

ACM Reference Format:
Theophilus A. Benson. 2019. In-Network Compute: Considered Armed and Dangerous. In *Workshop on Hot Topics in Operating Systems (HotOS '19), May 13–15, 2019, Bertinoro, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3317550.3321436

1 Introduction

The last few decades have seen a significant debate over the use and design of in-network functionality (NF). Until recently, much of this in-network functionality has been at layer-7, e.g., proxies. With the recent emergence of programmable data planes [7, 49], the functionality being placed into the network has shifted from pure layer-7 networking to include computing functionality, i.e., *in-network compute* (iCF).

A key defining property of iCF is a blurring of the traditional division between computing and networking infrastructures. In Table 1, we present a representative list of the functionality being developed for programmable data planes.

Functions	**Type**	**Mode**
Caching [31, 32, 32, 36, 43] e.g., NetCache [32]	iCF	Offload
NFV [24, 45, 46, 61] e.g., SilkRoad [46]	NF	Transparent
Consensus [16, 17, 27, 37, 38] e.g., P4xos [16]	iCF	Transparent
ML/AI, e.g., [53]	iCF	Offload
Stream processing [13, 29, 54] e.g., DAIET [54]	iCF	Offload

Table 1. Taxonomy of functionality deployed on programmable dataplanes.

For many of these approaches, e.g., Daiet [54], a fraction of the application logic is rewritten and offloaded onto programmable data planes (PDPs) within the network (Offload mode in Table 1). [*] The code running on a PDP is called a PDP program and written in a PDP language, e.g., P4.

When combining networking and computing functionality onto a PDP, network operators must grapple with management issues, e.g., infrastructure updates, troubleshooting, and diagnosis, for both networking and computing infrastructures. Moreover, as adoption of iCFs grows in size and prevalence, we believe these issues will only worsen.

The principal position of this paper is that *rearchitecting server applications to cross administrative boundaries (network and compute) introduces a semantic gap in existing, otherwise silo-ed, management techniques and complicates infrastructure managment.* In particular, while iCFs enables server applications to cross logical boundaries between computing and networking infrastructure, many current management abstractions, algorithms, and frameworks are limited in scope to either the networking or computing domains, and thus lack sufficient information and data plane primitives to effectively manage applications that cross these boundaries. For example, to install an iCF on a network device, the network requires both placement information (e.g., location) and network virtualization primitives that enforce isolation.

Naively merging existing management tools to enable cross-layer management is unscalable; creating new cross-layer tools is undesirable, since such tools would disregard practical administrative boundaries (i.e., NetOps versus DevOps). Instead of merging or rewriting existing tools, we

HotOS '19, May 13–15, 2019, Bertinoro, Italy

ACM ISBN 978-1-4503-6727-1/19/05...$15.00
https://doi.org/10.1145/3317550.3321436

[*]Note that while some in-network compute functionality require application modifications, some do not (i.e., transparent mode in Table 1).

argue that we should augment existing management techniques with abstractions and algorithms to enable a cross-layer exchange of information, allowing existing methods to more effectively manage the network.

To this end, we propose a system, Harmony, that facilitates cross-layer management of iCFs by introducing abstractions for data exchange between networking and computing management systems and a set of supporting data plane primitives. Harmony is inspired by recent work on cloud management [2, 3, 6, 51] while addressing issues specific to programmable data planes.

In particular, we identify the following key management challenges and introduce data plane primitives and information sharing abstrations to address them. Harmony must (1) simultaneously and dynamically support multiple iCFs, (2) allow for the transfer of placement information, (3) enable on-demand reaction to management events, and (4) disentangle diagnosis complexity. Programmable data planes support a broad range of functionality (Table 1). Thus, a key challenge lies in simultaneously ensuring generality and practicality; we illustrate the generality and practicality of our abstractions with several classes of emerging in-network compute applications.

2 Background

2.1 Managing Distributed Infrastructures

Many distributed applications are required to survive server failures and to scale in response to application workload fluctuations. We use the term orchestrator to broadly classify the processes that manage these applications (i.e., the process that allocates resources, monitors the tasks for failures, or scales them in response to workload fluctuations). For containers and microservices, Kubernetes is a popular orchestrator. For big data processing systems, e.g., MapReduce, the job manager is the orchestrator. On the networking side, a controller encapsulates the control logic that manages distributed switches (i.e., the process that installs network paths, configures/installs NFs, and reacts to failures).

2.2 Overview of NF

In Table 1, we present a taxonomy of recent approaches classified by the type of functionality, i.e., iCF or NF, and the mode of operation, i.e., offloaded (requires server application rewrite) or transparent (server application is transparent). We note that NFs do not require modifications to applications because these functions are naturally distinct from applications. An iCF, in contrast, may or may not require application modifications depending on the functionality being embedded into the network. Next, we highlight the salient features of iCFs by describing two complementary approaches.

- **NetCache [32]:** NetCache improves the performance of traditional key-value stores by using programmable data-planes as a caching layer between application servers and the key-value stores. The NetCache iCF is designed to run on ToR switches, which, traffic destined to the key-value servers will naturally traverse. The memory requirement for the PDP program is a function of the number of cache servers at the ToR. Failure of a programmable device impacts performance but not correctness because the network maintains only soft state.

  To effectively utilize NetCache, the orchestrator much be able to configure the PDP program based on runtime knowledge of the location of key-value stores and the number of key-value servers at each ToR. Without orchestrator involvement, the controller is, thus, unable to appropriately place or configure NetCache.
- **DAIET [54]:** DAIET offloads reducer tasks into the network. The network is used to aggregate results and, in turn, reduce latencies and improve throughput. The failure of a DAIET element significantly impacts the map-reduce job, because a DAIET iCF stores persistent state, the job may need to be re-run if intermediate data is lost. This problem is analogous to the failure of an intermediate server node in MapReduce which forces the job to recalculate lost components [33].

  To effectively utilize DAIET, the orchestrator must inform the controller of the location of the different mapper tasks as well as the reducer PDP program(s) to run. Given this information, the controller can appropriately place and route traffic through the reducer iCF (s).

Takeaway: As illustrated above, the use of programmable data planes as compute units naturally requires some coordination between the orchestrator and controller. The type and level of coordination is a function of the type of computation being embedded into the network. For example, placement information is required for resource allocations and workload information may be required for configuration.

Thus, migration of traditional computing tasks into the network complicates the functionality of an orchestrator because the orchestrator must now interact with and manage both the conventional application processes on servers and the iCFs within the data plane.

3 Management Challenges

In this section, we discuss several emerging trends in computing and discuss the management implications of a tighter coupling between the orchestrator and controller.

Challenge 1: Networking as a Utility. Emerging virtualization trends are slowly eliminating the explicit dependencies between the applications and the network. Two particular trends stand-out, microservice service-mesh [10] and

IaaS-overlay networking [15, 20, 35]. In both trends, the network is abstracted into one "big switch" with overlay tunnels used to transport traffic between endpoints.

Impact: A consequence of this trend is that the orchestrators are unaware of the underlying network and make many decisions in a network agnostic manner.

Requirement: The move to push compute functionality into the network will require orchestrators to either understand the topology to perform placements or to delegate placement functionality to controllers.

Challenge 2: Maintenance Activities. Most operational data centers require constant maintenance, e.g., hardware maintenance or firmware upgrades, which often needs rebooting one or more switches or taking switches offline. Today, elaborate algorithms [41, 62] are required to schedule maintenance activities in a manner that minimizes disruption of the network. These intelligent algorithms shield applications from maintenance activities by ensuring that certain invariants hold during each maintenance window. The move to offload compute into the network magnifies the disruption caused by these maintenance activities.

Impact: To effectively operate during maintenance windows, iCF must either be migrated off devices before a reboot (or shutdown) or the application must be redesigned to explicitly handle maintenance activities within the network. We argue that the least disruptive option, in-network support for migration, should be explored. While this option aligns with an application's expectation, it introduces new challenges.

Requirement: Migrating iCF requires coordination between the controller and the orchestrator to ensure that: (1) migration preserves the placement constraints, (2) the appropriate state is effectively migrated, and (3) migration is fast, correct, and incurs low overhead.

Challenge 3: Brittle Virtualization Abstractions. Central to a thriving iCF ecosystem is the ability to run multiple functions on a single device. For example, a ToR switch running a NetCache iCF will need to additionally run an ECMP NF to route and balance traffic.

Impact: Today, programmable data plane devices lack virtualization primitives, i.e., they can support only one program. Thus, there are no appropriate techniques for ensuring isolation and resource management between the different programs colocated on a device.

Requirement: The design of a virtualization and composition primitive for programmable data planes is a fundamentally distinct problem from the composition and virtualization of OpenFlow rules [5, 30, 48] because PDP programs [8] include complex structures (i.e., control flow and parser graphs) that must be appropriately virtualized. Moreover, both the PDP hardware and language constructs, e.g., Tofino and P4, provide insufficient primitives for building a virtualization framework.

Challenge 4: Insufficient Diagnosis Abstractions. Data center diagnosis and troubleshooting falls into two broad classes. First, distributed tracing [14, 21, 44, 56] traces RPC calls between functions to localize the offending component(s). Second, cross-layer diagnosis [4, 23] aims to localize the rootcause of a problem to either the network or end-host after which more appropriate techniques can be used.

Impact: The introduction of in-network compute complicates both directions: first, programmable devices do not provide a rich enough abstraction to support traditional tracing. Second, cross-layer diagnosis assumes a clear division between the network and the end host; however, in-network compute blurs this distinction.

Requirement: Fortunately, emerging programmable data planes provide a primitive, network telemetry, for introspecting on internal device state. We argue that a natural first step is to extend existing distributed tracing to incorporate network telemetry [11]. However, combining network telemetry with distributed tracing requires rethinking assumptions and limitations about tracing. While network telemetry has proved useful for tracking packet trajectories, distributed tracing requires capturing additional information and maintaining potentially complex and stateful data structures.

4 Design of Harmony

The insight behind Harmony is that the management challenges introduced by the in-network compute paradigm arise for two reasons: (1) a silo between the compute and the network resource managers (i.e., controller and orchestrator) which prevents the exchange of information necessary for coordination and orchestration of resources, and (2) a lack of appropriate abstractions within the PDP ecosystem to support virtualization. To address these two broad challenges, we present Harmony, a system that (1) introduces new *mechanisms* into the data plane for virtualizing the network (i.e., supporting isolation and providing QoS), and (2) provides new abstractions that allow the controller to capture (or expose) *policy information* from the orchestrator to efficiently manage and allocate resources.

Figure 1 presents an overview of Harmony. They key mechanisms and abstractions provided by Harmony are incorporated in the following components (orange boxes in Figure 1). First, a lightweight hardware virtualization layer, **PDPVisor**, that enables efficient composition of multiple PDP programs through source code merging and provides provable isolation and safety through compiler optimizations; the PDPVisor also introduces a primitive for augmenting iCF to automatically generate distributed tracing data. Second, a set of management interfaces: **NWLeases**, a push-based interface, which allows the orchestrator to specify and push placement requirements and configuration as an annotated graph; and **NWCallBacks**, a pull-based interface, which allows the controller to inform the orchestrator of

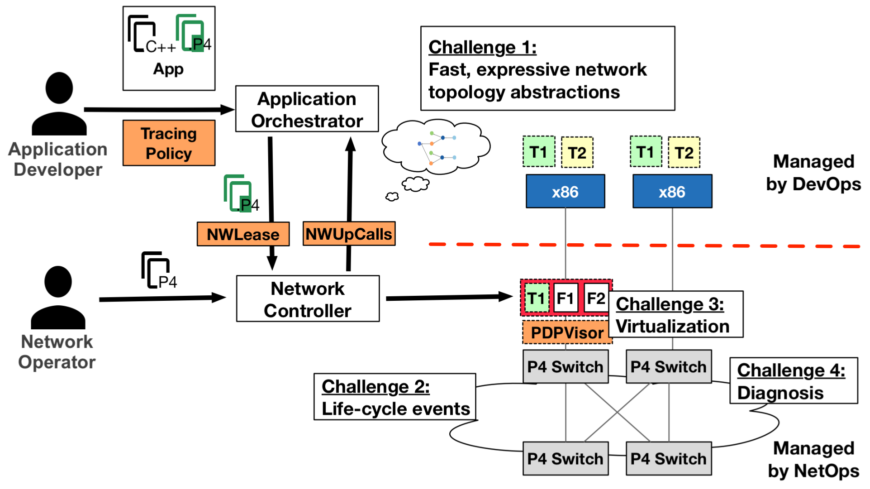

Figure 1. *Harmony: high level vision.*

network dynamics and allows the orchestrator to react appropriately, e.g., perform state management and cleanup.

4.1 Bridging the Semantic Gap

Placement and configuration are two key pieces of information required to ensure that iCFs function correctly and appropriately accelerate applications. For configuration, we anticipate that this can be expressed in a configuration file.

For placement information, there are two broad directions. First, Harmony could expose network information to the orchestrator to decide placement. Network information can be exposed directly, e.g., as a graph of the physical topology [39], or abstractly, e.g., via a query interface [2, 55]. This direction has a few drawbacks: it introduces scalability challenges, raises potential privacy/security issues when the network/compute are run by different organizations, and complicates networking resource allocation because two different entities are performing allocation/management of the same network resources. Second, Harmony could expose application requirements [5, 6, 28] to the network for the controller to perform placement. This direction centralizes resource allocation logic and allows for more efficient placement decisions, but requires a flexible and expressible interface or a domain-specific language (DSL) to effectively capture varied application requirements.

While more challenging, we opt for the latter direction because it provides one entity, the controller, with global visibility and control over resource allocations. This enables the controller to explore globally efficient placement decisions that maximize use of network resources.

Existing interfaces [6, 28] for capturing application placement requirements are ineffective because they are targetted towards capturing placement information that impacts VM performance relative to other VMs. We require interfaces that allow us instead to capture VM/endhost performance relative to the choice of network PDP devices used for iCFs. For example, abstractions in Oktopus [6] allow tenants to specify placement constraints as bandwidth requirements, where as existing iCFs, e.g., NetCache or SilkRoad, will require precise control over placement (e.g., placement on a directly connected ToR). Additionally, these abstractions do not enable placement of a reducer relative to the location of specific mappers. Extending these existing interfaces requires significant modifications to accept graph-based representations.

Crafting Expressive Abstractions: The orchestrator informs the controller of placement information as a graph that indicates the placement of iCFs relative to pre-allocated VMs or server applications. The graph consists of two types of nodes: iCF (indicated as circles in Figure 2) and servers (indicated as boxes in Figure 2). The iCF nodes can be annotated as ToR or non-ToR to indicate strict placement constraints, e.g., in Figure 2 (b), the nodes are annotated with a T for ToR. Edges in the graph capture relative placement constraints and indicate routing constraints. For example, to specify that a reducer's placement is relative to the server(s) hosting mappers, the reducer node should have links directly to the mapper nodes (in Figure 2 (c), the links represent constraints between two mappers (CF5 and CF6) and a reducer iCF (iCF7)).

NWLeases: More formally, our abstraction, NWLeases, captures the PDP program (source code), the placement constraints (as a graph), resource requirements (as configurations), and the requested time length of the lease. The abstraction (Table 2) provides mechanisms to modify the NWLease, renew it, or delete it. In designing our abstraction, we aim to minimize the complexity of the controller. In particular, we reduce the state and functionality at the controller by expressing resource allocation as leases, freeing the controller from tracking application availability.

The orchestrator creates and modifies the NWLeases as the deployed application is modified. Since many applications can operate without their iCFs, a lease can be created at any

Operation	Returns	Description
new NWLease()	LeaseID	creates a new lease
ConfigLease(LeaseID, Program, placement parameters, PlacementGraph, Config)	Boolean	augments the PDP program, or configuration of a lease
ReNewLease(LeaseID)	Boolean	Renews the NWLease
DeleteLease(LeaseID)	Boolean	Deprovisions the iCFs and their associated paths

Table 2. NWLease API: supported calls.

point. Successful creation of the NWLease may reroute traffic to utilize the newly placed iCFs. Additionally, modifications to a NWLease may also lead to traffic rerouting if placement decisions are modified.

Enforcing Placement and Configuration: Given the above abstraction, the controller can solve a placement problem which tries to place different iCFs (and NFs) subject to placement and resource configuration constraints from the orchestrator, physical device constraints placed by the PDP devices, and operator specified objectives. Formulating the placement problem and providing practical solutions is out of the scope of this work; however, we note that several directions [12, 19, 52, 66] are being made along these lines within the context of NFV-placement — we believe such work can be easily extended to our domain.

4.2 Optimizing for Network Changes

The NWLeases abstraction enables the controller to transparently react to maintenance activities (e.g., upgrades) by creating a new placement and reallocating iCFs. However, reallocating iCFs can impact application performance. Next, we discuss an interface that optimizes performance during network dynamics by enabling the orchestrator to react to these network events directly.

We argue that while initial provisioning and configuration are amenable to a push-based interface, for real-time reaction to network changes, a pull-based approach is required. The controller pulls information from the orchestrator to enable efficient reaction to network changes. We envision that during network changes, the controller alerts the orchestrator of the change, provides the orchestrator with an opportunity to perform cleanup or bookkeeping activities, and pulls information from the orchestrator to inform placement or migration.

Our interface is inspired by container management primitives, i.e., Kubernetes' lifecycle hooks [63]. These "upcalls" allow hypervisors to run pre-specified code in containers during lifecycle events (e.g., container creation or deletion). In our context, the orchestrator registers for events concerning their iCFs. Given these registered handlers, the controller makes appropriate upcalls. We envision that Harmony will support events including migration (no state is lost but potential performance penalties may be incurred during migration), reallocation (migration due to placement optimization), and shutdown (an event that results in state loss).

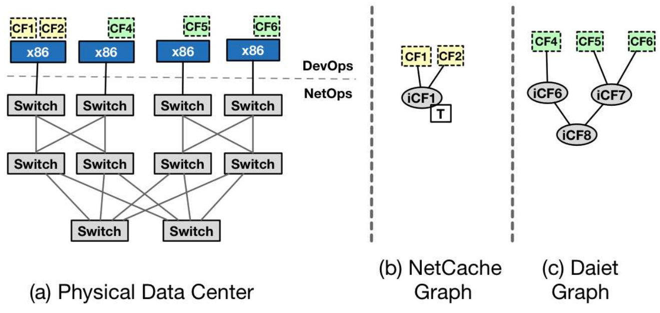

Figure 2. *(a) Physical data center with NetCache (yellow) and DAIET (green), (b) NWLeases for NetCache , and (c) NWLeases for DAIET.*

4.3 Composing In-Network Functionality

As discussed in Section 3, the PDP hardware and software toolchain do not support running multiple P4 programs concurrently, either on programmable switches or smart NICs. Existing approaches [25, 65] to virtualize programmable dataplanes explore emulated virtualization; unfortunately, the emulation introduces significant overheads (e.g., as high as 20X [67]). Furthermore, PDP devices do not include primitives required to support more efficient virtualization approach, e.g., paravirtualization. Instead, we propose a non-conventional approach: Lightweight virtualization through source code merging where virtualization is achieved by merging all the PDP programs into one, which retains the functionality of the original PDP programs. Our source code merging-based approach to virtualization introduces the following key challenges:

- First, we need to ensure that, when merged, the final program retains the properties of each of the original NFs and iCFs while simultaneously providing isolation and

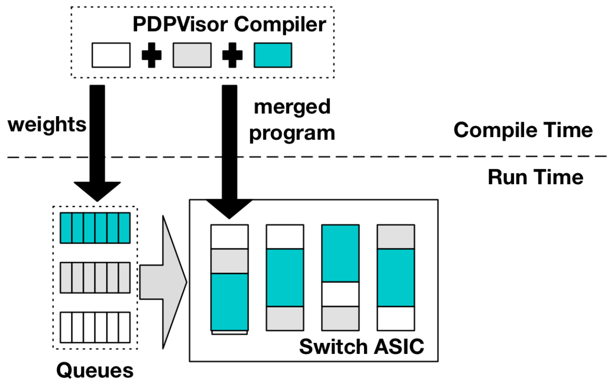

Figure 3. *PDPVisor: overview.*

enforcing resource quotas. During compilation, PDP programs are represented as directed acyclic graphs (DAGs), so we can view the merge as a DAG merging problem and place appropriate constraints, e.g., the merged graph must be a DAG, to ensure that the merged DAG ensures isolation between the different functions on a PDP device. Another benefit of DAG-based representations is that we can easily enforce resource allocation constraints on nodes within the graph. Moreover, we recently showed that the program merging problem is equivalent to the weighted maximum independent set (WMIS) graph problem, which, while NP-hard, can often be solved in polynomial time with a greedy heuristic [67].

- Second, at runtime, we need to enforce quality of service guarantees, e.g., an iCF should be allocated 60% of a device's processing cycles. Fortunately, the number cycles required to process a packet can be determined apriori by analyzing the merged PDP program. Thus, we can control the cycles allocated to each NF and iCF by allocating queues and token buckets [57].
- Third, updating the PDP program on a device introduces several seconds of disruption because the device needs to reboot to maintain state consistency [59] (similar to OS updates). Thus, provisioning or de-provisioning NFs and iCFs will introduce disruptions because the merged program will need to be updated. An open challenge lies in understanding how to perform headless updates to the data plane without impacting availability.

Figure 3 presents an overview of PDPVisor. The main component is the compiler, which takes as input N PDP programs and creates as output one PDP program with associated queue weights. The use of the compiler enables Harmony to analyze the programs, ensure isolation and allocate resources according to predefined resource sharing constraints. Resource allocations for ASIC processing cycles are enforced with queues and token buckets; allocations for switch memory are enforced by limiting the number of table entries allocated.

4.4 In-Network Tracing

Extending tracing [21, 44] into the network requires introducing primitives into the data plane to capture contextual information (i.e., IDs, tags, and timings) and policies to determine when to capture and report them. While this logic is easy to implement on servers, limited programming models make introducing primitives and policies to PDPs nontrivial. Additionally, while emerging in-network telemetry techniques [11, 26, 40] provide packet trajectories, i.e., the path taken by a packet, these techniques do not include some of the broader contextual information required for distributed tracing. For example, although P4 makes it easy to capture information for a specific RPC request, understanding changes and evolution requires maintaining state within the switch, a more challenging task.

Language model (i.e., P4) and programmable switch design (e.g., Tofino) introduce limitations on how state can be stored (e.g., in array registers) and accessed (e.g., sequential and once-per-packet). Our main innovation is to design stateful data structures that efficiently store contextual information required to effectively create distributed traces while respecting hardware constraints. For example, designing sketch-based algorithms for compactly sampling and capturing per-application contexts.

5 Related Work

Existing work focuses largely on demonstrating the flexibility of modern programmable dataplanes to support a broader range of computation [17, 29, 31, 32, 36, 43, 53, 54] or network functionality [24, 45, 47, 58], or on developing abstractions and algorithms to improve the functionality [9, 34, 50], flexibility [25, 65], efficiency [1], and management [18, 22, 42, 60, 64] of existing PDPs.

Our work differs from existing PDP management approaches [18, 64], in that we aim to support integration of the network control plane with the compute control plane. Existing management directions will benefit from the information that our work exposes.

While our work is inspired by a rich body of work on cloud management [2, 3, 6, 51] our work revisits these questions within the context of in-network compute and explores the practical challenges introduced by programmable data planes: limited resources and a significantly constrained language model.

6 Conclusion

Designing a network management plane that explicitly embraces in-network computation introduces many interesting challenges. In addition to the diagnosis, maintenance, and management challenges discussed here, in-network compute also complicates security and reliability. Our goal is to begin the discussion on a subset of these management issues by shedding light on a few key issues. As part of future work, we will explore other directions including security and reliability.

7 Acknowledgments

We thank the anonymous reviewers and our shepherd, Eddie Kohler, for their helpful comments. This work was supported by NSF award CNS-1749785.

References

[1] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, and A. Akella. P5: Policy-driven Optimization of P4 Pipeline. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 136–142, New York, NY, USA, 2017. ACM.

[2] A. Agache, M. Ionescu, and C. Raiciu. CloudTalk: Enabling Distributed Application Optimisations in Public Clouds. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 605–619, New York, NY, USA, 2017. ACM.

[3] N. Amit and M. Wei. The Design and Implementation of Hyperupcalls. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 97–112, Boston, MA, 2018. USENIX Association.

[4] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 440–453, New York, NY, USA, 2016. ACM.

[5] A. AuYoung, Y. Ma, S. Banerjee, J. Lee, P. Sharma, Y. Turner, C. Liang, and J. C. Mogul. Democratic Resolution of Resource Conflicts Between SDN Control Programs. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 391–402, New York, NY, USA, 2014. ACM.

[6] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 242–253, New York, NY, USA, 2011. ACM.

[7] Barefoot Networks. Barefoot Tofino, 2018.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[9] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. *SIGCOMM Comput. Commun. Rev.*, 43(4):99–110, Aug. 2013.

[10] B. Burns and D. Oppenheimer. Design Patterns for Container-based Distributed Systems. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, 2016. USENIX Association.

[11] P. B. Changhoon Kim and E. Doe. In-band Network Telemetry (INT), 2016. [Online; accessed 14-July-2016].

[12] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou. Multi-Commodity Flow with In-Network Processing. *CoRR*, abs/1802.09118, 2018.

[13] L. Chen, G. Chen, J. Lingys, and K. Chen. Programmable Switch as a Parallel Computing Device. *CoRR*, abs/1803.01491, 2018.

[14] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch. The Mystery Machine: End-to-end performance analysis of large-scale Internet services. In *Proceedings of the 11th symposium on Operating Systems Design and Implementation*, 2014.

[15] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E. C. Zermeno, E. Rubow, J. A. Docauer, J. Alpert, J. Ai, J. Olson, K. DeCabooter, M. de Kruijf, N. Hua, N. Lewis, N. Kasinadhuni, R. Crepaldi, S. Krishnan, S. Venkata, Y. Richter, U. Naik, and A. Vahdat. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 373–387, Renton, WA, 2018. USENIX Association.

[16] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, M. Canini, N. Zilberman, F. Pedone, and R. Soulé. P4xos: Consensus as a Network Service. Technical Report 2018/01, University of Lugano, May 2018.

[17] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé. NetPaxos: Consensus at Network Speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 5:1–5:7, New York, NY, USA, 2015. ACM.

[18] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon. Whippersnapper: A P4 Language Benchmark Suite. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 95–101, New York, NY, USA, 2017. ACM.

[19] G. Even, M. Rost, and S. Schmid. An Approximation Algorithm for Path Computation and Function Placement in SDNs. In *Proc. 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2016.

[20] D. Firestone. VFP: A virtual switch platform for host SDN in the public cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 315–328, Boston, MA, 2017. USENIX Association.

[21] R. Fonseca, G. Porter, R. H. Katz, and S. Shenker. X-Trace: A Pervasive Network Tracing Framework. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, Cambridge, MA, 2007. USENIX Association.

[22] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, and M. Barcellos. Uncovering Bugs in P4 Programs with Assertion-based Verification. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 4:1–4:7, New York, NY, USA, 2018. ACM.

[23] M. Ghasemi, T. Benson, and J. Rexford. Dapper: Data Plane Performance Diagnosis of TCP. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 61–74, New York, NY, USA, 2017. ACM.

[24] H. Giesen, L. Shi, J. Sonchack, A. Chelluri, N. Prabhu, N. Sultana, L. Kant, A. J. McAuley, A. Poylisher, A. DeHon, and B. T. Loo. In-network Computing to the Rescue of Faulty Links. In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, pages 1–6, New York, NY, USA, 2018. ACM.

[25] D. Hancock and J. van der Merwe. HyPer4: Using P4 to Virtualize the Programmable Data Plane. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 35–49, New York, NY, USA, 2016. ACM.

[26] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 71–85, Seattle, WA, 2014. USENIX Association.

[27] Z. István, D. Sidler, G. Alonso, and M. Vukolic. Consensus in a Box: Inexpensive Coordination in Hardware. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 425–438, Santa Clara, CA, 2016. USENIX Association.

[28] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Bridging the Tenant-provider Gap in Cloud Services. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 10:1–10:14, New York, NY, USA, 2012. ACM.

[29] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé. Life in the Fast Lane: A Line-Rate Linear Road. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 10:1–10:7, New York, NY, USA, 2018. ACM.

[30] X. Jin, J. Gossels, J. Rexford, and D. Walker. CoVisor: A Compositional Hypervisor for Software-defined Networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 87–101, Berkeley, CA, USA, 2015. USENIX Association.

[31] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, 2018. USENIX Association.

[32] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 121–136, New York, NY, USA, 2017. ACM.

[33] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. On Availability of Intermediate Data in Cloud Computations. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09, pages 6–6, Berkeley, CA, USA, 2009. USENIX Association.

[34] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, and K. Rothermel. P4CEP: Towards In-Network Complex Event Processing. In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, pages 33–38, New York, NY, USA, 2018. ACM.

[35] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet,

S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network Virtualization in Multi-tenant Datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, Seattle, WA, 2014. USENIX Association.

[36] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 137–152, New York, NY, USA, 2017. ACM.

[37] J. Li, E. Michael, and D. R. K. Ports. Eris: Coordination-Free Consistent Transactions Using In-Network Concurrency Control. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 104–120, New York, NY, USA, 2017. ACM.

[38] J. Li, E. Michael, N. K. Sharma, A. Szekeres, and D. R. K. Ports. Just Say No to Paxos Overhead: Replacing Consensus with Network Ordering. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 467–483, Berkeley, CA, USA, 2016. USENIX Association.

[39] Y. Li, D. Wei, X. Chen, Z. Song, R. Wu, Y. Li, X. Jin, and W. Xu. DumbNet: A Smart Data Center Network Fabric with Dumb Switches. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 9:1–9:13, New York, NY, USA, 2018. ACM.

[40] J. Liang, J. Bi, Y. Zhou, and C. Zhang. In-band Network Function Telemetry. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, pages 42–44, New York, NY, USA, 2018. ACM.

[41] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz. zUpdate: Updating Data Center Networks with Zero Loss. *SIGCOMM Comput. Commun. Rev.*, 43(4):411–422, Aug. 2013.

[42] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, and N. Foster. P4V: Practical Verification for Programmable Data Planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 490–503, New York, NY, USA, 2018. ACM.

[43] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya. IncBricks: Toward In-Network Computation with an In-Network Cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 795–809, New York, NY, USA, 2017. ACM.

[44] J. Mace and R. Fonseca. Universal context propagation for distributed system instrumentation. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 8:1–8:18, New York, NY, USA, 2018. ACM.

[45] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares, and E. Rojas. ARP-P4: A Hybrid ARP-Path/P4Runtime Switch. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 438–439, Sep. 2018.

[46] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 15–28, New York, NY, USA, 2017. ACM.

[47] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 15–28, New York, NY, USA, 2017. ACM.

[48] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing Software Defined Networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 1–13, Lombard, IL, 2013. USENIX Association.

[49] R. Ozdag. Intel® Ethernet Switch FM6000 Series-Software Defined Networking. 2012.

[50] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson. Floem: A Programming System for NIC-accelerated Network Applications. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, pages 663–679, Berkeley, CA, USA, 2018. USENIX Association.

[51] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[52] G. Sallam, G. R. Gupta, B. Li, and B. Ji. Shortest Path and Maximum Flow Problems Under Service Function Chaining Constraints. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 2132–2140, April 2018.

[53] D. Sanvito, G. Siracusano, and R. Bifulco. Can the Network Be the AI Accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, pages 20–25, New York, NY, USA, 2018. ACM.

[54] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis. In-Network Computation is a Dumb Idea Whose Time Has Come. In *Proceedings of the Sixteenth ACM Workshop on Hot Topics in Networks*, 2017.

[55] J. Seedorf and E. Burger. Application-Layer Traffic Optimization (ALTO) Problem Statement. RFC 5693, Oct. 2009.

[56] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. *Google research*, 2010.

[57] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable Packet Scheduling at Line Rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 44–57, New York, NY, USA, 2016. ACM.

[58] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith. Turboflow: Information Rich Flow Record Generation on Commodity Switches. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 11:1–11:16, New York, NY, USA, 2018. ACM.

[59] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith. Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 823–835, Boston, MA, 2018. USENIX Association.

[60] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu. Debugging P4 Programs with Vera. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 518–532, New York, NY, USA, 2018. ACM.

[61] N. Sultana, S. Galea, D. Greaves, M. Wojcik, J. Shipton, R. Clegg, L. Mai, P. Bressana, R. Soulé, R. Mortier, P. Costa, P. Pietzuch, J. Crowcroft, A. W. Moore, and N. Zilberman. Emu: Rapid Prototyping of Networking Services. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 459–471, Santa Clara, CA, 2017. USENIX Association.

[62] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin. A Network-state Management Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 563–574, New York, NY, USA, 2014. ACM.

[63] The Kubernetes Authors. Container Lifecycle Hooks, 2019.

[64] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon. P4FPGA: A Rapid Prototyping Framework for P4. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 122–135, New York, NY, USA, 2017. ACM.

[65] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, and J. Wu. HyperV: A High Performance Hypervisor for Virtualization of the Programmable Data Plane. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, July 2017.

[66] J. Zhang, A. Sinha, J. Llorca, A. M. Tulino, and E. Modiano. Optimal Control of Distributed Computing Networks with Mixed-Cast Traffic Flows. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1880–1888, 2018.

[67] P. Zheng, T. Benson, and C. Hu. P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, pages 98–111, New York, NY, USA, 2018. ACM.