On Programmable Networking Evolution

Pravein Govindan Kannan Mun Choon Chan
School of Computing
National University of Singapore

Abstract

In this paper, we present a short survey of programmable networks starting from pre-SDN (Software Defined Networking) efforts to the more recent programmable data-plane. In particular, we highlight the benefits of data-plane programmability and present some of the challenges and future research directions.

1 Introduction

Computer networks play a critical role in modern technology. For a long time, computer networks were hardware based solutions with rigid and proprietary applications and devices. The minimal functionality that was assumed from the network (routers) is simply best-effort packet forwarding. However, treating routers and switches as monolithic black boxes with limited capability poses an enormous challenge for the network administrator. This challenge derives from the need to understand, maintain and analyze a complex network consisting of a large number of networking devices.

One approach to address this challenge is to *open up* the networking devices to make them **more programmable**. Clearly, there are obvious questions that arise. First, what are the right programming models and abstractions? Second, what are the incentives for network equipment manufacturers to "open up the box"? An open network platform which gives the users more power and flexibility may not make good business sense. The second question is beyond the scope of this paper. In this paper, we will discuss only the first question.

The rest of the paper will be organized as follow. In §2, we briefly present efforts made to enhance network programmability till roughly mid 2000s. These efforts include various proposals from academic, standard bodies and industries. In §3, we present OpenFlow, which starts a new era of programmable networks termed as Software Defined Networking (SDN) with its data/control separation and logically centralized control plane. In §4, we present the recent developments in programmable networks in the form of programmable data plane. In §5, we present some interesting applications that have been enabled by these recent developments. Finally, in §6, we discuss challenges and possible future research directions.

2 Pre-SDN Programmable Model

While network programmability has attracted a lot of interest and, promises much practical impact in recent years, many of the concepts that the current generations of programmable networks are not new and have been proposed in earlier work.

Mid 1990s saw the development of active networks [22, 80]. There are two forms of active networking: 1) capsule model and, 2) switch/router model. In the capsule model, programs can be embedded in packets to deploy new services in the network. However, the capsule model rises lots of security concerns as it is possible for malicious end-host to infect a router and hence the network. The other form is the switch/router model, whereby the switch/router can be programmed to perform different tasks. This is conceptually similar to the data-plane programmability model to be discussed in §4.

The concept of an open programmable network was also proposed in works [24, 49] in the late 90s. The idea was to provide a set of application programming interfaces (APIs) that abstract network resources and services. By providing access to the network hardware via open, programmable network interfaces, new services, in particular, multimedia that require QoS guarantees can be built on top of these APIs through a distributed programming environment.

Another similar effort is the IETF General Switch Management Protocol (gsmp) [7] working group that was active from 1999 to 2003. The gsmp working group aimed to provide an interface that can be used to separate the data forwarder from the routing and other control-plane protocols. One of the objective of this separation is that it will allow easier addition of network services.

A more recent effort is the IETF Forwarding and Control Element Separation (ForCES) [6] working group (2001 - 2015) which also looks at the separation of control-plane and data-plane. The ForCES base protocol (RFC5810) is used to maintain the communication channel between the control element (CE) and forwarding element (FE). SoftRouter [47] is one of the earlier proposal to use logically centralized and separate control elements to control multiple forwarding elements. It uses the ForCES protocol for communication between the CEs and FEs.

Unfortunately, the above proposals were not widely adopted. Besides the fact that it is difficult to get vendor support, these ideas also received pushback because exposing the underlying hardware through API, a more complex control-plane and logically centralized route control could

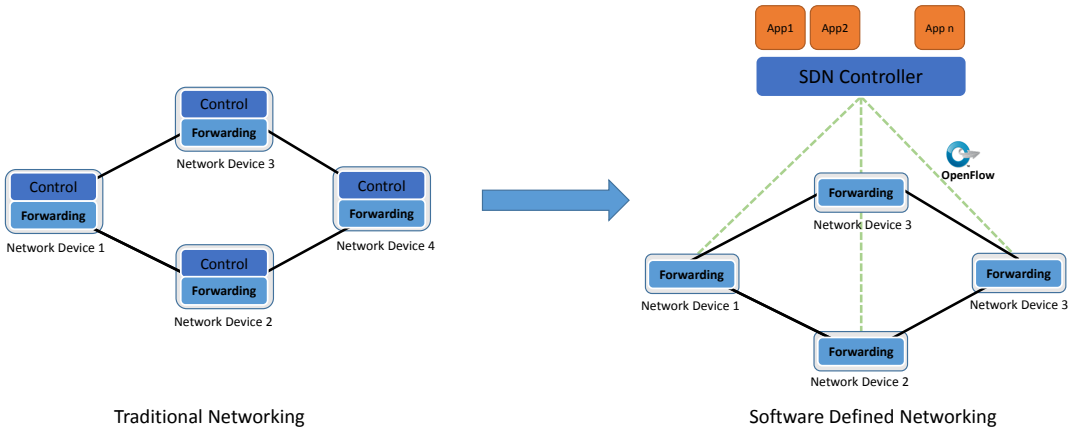

Figure 1: Software Defined Networking model compared to traditional networking

be argued as going against the simple/dumb network design philosophy of the internet that prefers a minimalist approach [28].

3 Sofware Defined Networking in the Control-Plane

In spite of the previous difficulties in incorporating programmable networking ideas in actual network deployment, programmable networks have recently enjoyed much more success in the form of Software Defined Networking (SDN). The success of SDN has created a fundamental change in how we manage or program networks in the following ways: 1) de-coupling the control-plane and data-plane in the switches, 2) logically centralized programmable control-plane and, 3) the development and introduction of programmable data-plane. We will look into aspect of (1) and (2) in this section and (3) in §4.

1. Decoupling Control and Data-plane: Proprietary networking devices have for long coupled control-plane and data-plane with proprietary solutions. Control-plane is the intelligence layer, which implements protocols like OSPF, BGP, Multicast, Traffic Engineering, etc. Data-plane performs the high-speed packet forwarding based on the rules and policies programmed by the control-plane. Each network device is an independent entity that needs to be managed separately in the network. Also, each network device contains complex functionality baked in, making it very complex to configure, debug and manage. SDN decouples the control/data-plane and provides an API for the control-plane to program the data-plane in the form of Match-Action rules. OpenFlow [62] provides this API as an open standard interface for the network operators to program the network devices. Many switch vendors such as Cisco, Juniper, HP, Arista, etc. support OpenFlow API to allow their devices to be programmed.

2. Logically Centralized Programmable Control-plane: Apart from decoupling control and data-plane, the control-plane is a logically centralized entity (a.k.a controller) as shown in Figure 1. The Controller being centralized, can see the network view, and provide a network-level abstraction for the applications to be developed over the controller. The controller, typically running on commodity servers, can install rules to network device, aggregate statistics from the network device to provide a One-Big-Switch [66] abstraction.

While the idea of control/data plane separation and logical centralized control may have already been proposed in previous work, SDN, in particular OpenFlow [62], is the first successful and widely deployed framework for network programmability. The popularity of OpenFlow may have been due to the ease of adoption by switch vendors using just software (OS) upgrades. Another strength of OpenFlow is the availability of many SDN controllers based on OpenFlow APIs targeting both academia (NOX [32], POX [15], Floodlight [5], Ryu [16], etc.) and industry (ONOS [12], OpenDayLight [13], etc.). OpenFlow APIs allow the SDN controllers to program rules in switches that match several fields on the packet, and take actions like forwarding, meter, queuing, filtering, etc. OpenFlow 1.0 started simple with matching on 12 fields (Ethernet, TCP/IPv4), while OpenFlow 1.5 supports matching on 44 fields [25].

OpenFlow was initially deployed in campus networks. Early commercial successes, such as Google's wide-area traffic-management system [38] and adoption by large telcos such as AT&T [1] has also boosted its attractiveness. The emergence of data center as the key workhouse in modern internet service provisioning also helped to make SDN deployment attractive. The data center network is a large network managed by a single entity. It demands advance feature to deal with the scale and can be managed centrally.

[1] https://www.onap.org/wp-content/uploads/sites/20/2019/07/35752846-0-Lumina-Orchestration-2.pdf

A more comprehensive survey for SDN up till early 2010 can be found in [63] and [67].

Much of the early vision for SDN focused on control-plane programmability. The next step is data-plane programmability.

4 Programmable Data-plane

While the control-plane programming allows a control program to modify the Match-Action rules in the data-plane, the matching fields and parsing logic are still pre-defined and relatively limited in scope. On the other hand, data-plane programming allows flexible parsing and matching on non-standard fields. This enables faster and easier network evolution. New protocols (or headers) addition which traditionally are big hardware upgrades that take 4-5 years, could be done by a software upgrade in a few months.

Programmable switches expose new datapath primitives that are previously not available, including:

1. Transactional Memory (SRAM) with stateful ALUs that can perform simple computations like add, subtract and approximate multiply/divide.

2. Precise timestamps at different stages (ingress/egress pipeline), useful in network telemetry like queue duration and depth.

3. Packet cloning/replication useful for flexible mirroring, reporting postcards and conditional multicast.

Recent networking architectures [10, 26] have enabled the following to be programmable:

- Parsing of a packet;
- Ingress/egress processing using flexible Match-Action tables;
- Stateful maintenance of network states using SRAMs (registers).

The model of programmable data-planes are standardized in the form of a Portable Switch Architecture (PSA) as shown in Figure 2.

Given all these additional capabilities and features, programming the data-plane is challenging considering that all processing *must* be performed at line-rate with no slow down in the packet processing pipeline. This constraint comes at the cost of the programming flexibility. For example, in the program to be run in the data-plane, there is no loop construct, no floating point computations and no complex computations like multiply/divide (only approximation possibly using bit-shifts). A memory unit is mapped to a single stage and it can be read/write only once in the single pass of the packet. In spite of these limitations, developing applications in data-plane provides an attractive option to achieve high processing throughput at low latency.

Data-plane programmable switches have gained major traction in the networking industry with programmable switching ASICs being released by major switch vendors such as BareFoot Tofino [20], Cavium [3], Intel Flexpipe [10], Cisco Nexus 3000 [4], Broadcomm Trident [2], Innovium [9] and Xilinx SDNet [17]. P4 programming language [25], which emerged from academic and industry efforts is currently widely used to program these devices.

5 Applications

In this section, we will be exploring the various applications developed in the network data-plane over the recent years. We categorize them into two main sections : 1) Network Monitoring and Debugging (§5.1) and 2) In-network Computing (§5.2).

5.1 Network Monitoring and Debugging

Traditional network monitoring has relied on sampling techniques like NetFlow [11], sFlow [18], SNMP [19] which collect flow information by sampling techniques. These techniques are useful in gathering coarse-grained statistics of the network traffic. However, they are not as effective for gathering fine-grained information such as flow-level statistics over short time-scales (in order of milliseconds). One limitation of existing approaches is that they need to reduce the amount of information exchange between control and data-plane. Hence, low sampling rates of 0.1% or less are used. Such sampling looses information and cannot capture an accurate picture of the network.

Recently, several research works have leveraged programmable networking to develop monitoring tools in the context of a single network device and also network-wide. All these works have been implemented in the data-plane using P4 running either on the BMv2 [2] simulator or hardware such as the Tofino switches. The commonality among these works is that they exploit the new programming capability in the data-plane to compute and store information that were not possible previously. We group them into two categories : 1) Local Monitoring: the monitoring context is local to a device. 2) Network-wide monitoring: The monitoring context is to provide network-wide statistics.

5.1.1 Local In-Network Monitoring

Count-Min Sketch [29], OpenSketch [85], SketchLearn [37] and ElasticSketch [83] design optimal data structures to store flow summary and features in the network data-plane. HashPipe [75] provides heavy-hitter detection in the network data-plane using a pipeline of hash-tables. BurstRadar [43] and Snappy [27] implement techniques in

[2] https://github.com/p4lang/behavioral-model

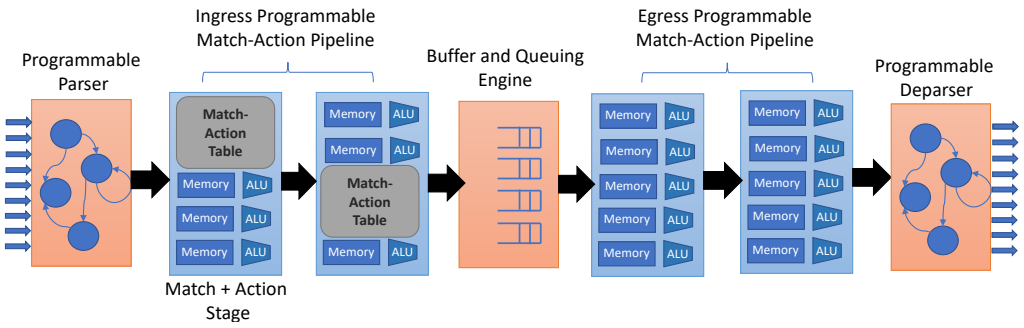

Figure 2: Data-plane programmability using Portable Switch Architecture [14]

the network data-plane to detect short-term events such as microbursts. BurstRadar [43] sends post-cards of the queue contents when microbursts occur to facilitate further debugging. Snappy [27] provides a probabilistic technique to identify the heavy flows in the network data-plane. Finally, *flow [77] implements mechanism to export telemetry information from switching ASICs in a flexible format.

5.1.2 Network-wide Monitoring and Telemetry

The ability to store states in the data-plane and to modify the packet header according to the application requirements have opened up many promising approaches for network-wide monitoring.

FlowRadar [55] provides per-flow counters for all the flows by encoding the flows and their counters with a small memory in the switches and performs aggregation of counters across switches to build network-wide information over a time scale of 10s of seconds. UnivMon [60] is similar in nature, but uses sketch data structure in the data-plane to compute aggregate statistics and provides more features like heavy-hitters, DDos victim detection, entropy estimation. Mozart [58] uses match-action rules to monitor flow statistics and the rules are placed to measure the flows at the right time subjected to the switch constraints. TurboFlow [76] is another recent proposal that uses data-plane capability to generate flexible flow-records from the network by aggregating records from switches.

While the previous approaches [55, 58, 60] measure statistics in the data-plane and extract the aggregated information from the control-plane, another approach is to convey the statistics using data packets instead of through the control-plane. For example, NetSight [34] generates packet-histories in the form of postcards for data packet and sent to end-hosts to build a debugger which can be used to debug a packet's history. Alternatively, In-band Network Telemetry (INT) [8] modifies the packet header to append telemetry data like queue depth and other meta. SwitchPointer [79] runs on INT and provides support for directory services and distributed debugging.

Finally, Sonata [33] and Marple [64] provide declarative interfaces to express queries for network-wide telemetry tasks by offloading the primitives of queries into network data-plane. Marple [64] additionally provides a scalable Key-Value store hardware primitive.

While these research works collect network-wide statistics, they do not present a synchronized or a consistent state of the network at fine time scales (microseconds). However, presenting a consistent picture of the network is important to debug network-wide faults like load imbalance, loops, misconfigurations, microbursts, etc.

Synchronized Debugging. OFRewind [81] enables coordinated replay of control-plane (OpenFlow [62]) events by recording controller messages and packets to diagnose OpenFlow configuration issues and bugs. Speedlight [84] performs synchronized (using PTP [21]) network snapshots using consistent snapshot algorithm in data-plane and control-plane clocks. It requires advance scheduling of snapshots every few milliseconds. One limitation of current works on synchronized debugging is that they are triggered by control-plane clocks which operate in the time scale of milliseconds. As a result, they are not effective in capturing ephemeral faults in the network that can occur in the order sub-milliseconds. Such ephemeral faults are more likely to occur as link speed is fast approach a few hundred Gbps.

5.2 In-network Computing

Programmable switches provide flexible access to SRAMs and additionally provide stateful computations. It enables certain computations to be offloaded from the end-hosts to the switches. In this section, we summarize the research on offloading computation to the network (termed In-network Computing). We broadly categorize works into areas: 1) Improving Network Performance, 2) In-Network Aggregation & Caching and, 3) Assisting (Accelerating) Distributed Systems.

Improving Network Performance. Several research works have leveraged programmable switches to enable the network to provide more than just a best-effort service. HULA [46] performs load balancing in the network by maintaining live tracking of congestion levels of all paths to a destination. NDP [35] presents a new data center protocol architecture to achieve high throughput and low latency of

network flows. Sharma et al. [73] implements traditional congestion control protocols like XCP [45] and RCP [78] at line-rate. AFQ [74] implements approximate fair-queuing using programmable switches. Elmo [72] enables scalable multicast in data-centers with very low traffic overhead. Blink [36] performs network fast path recovery in the data-plane by listening to TCP signals in the data-plane. Finally, SQR [69] performs in-network packet-loss recovery during link failures.

In-Network Aggregation & Caching. DAIET [70] was one of the earliest works to show-case the benefits of performing in-network aggregation using programmable data-plane. It performed aggregation of key-value pairs of big-data workloads and observed upto 89% reduction in traffic. Several recent works like IncBricks [57], NetCache [42], SwitchKV [54], KV-Direct [51], DistCache [59] and SwitchML [71] leverage programmable switches to cache key-value pairs to perform traffic reduction by caching hot key-value pairs at line-rate and additionally free up some computation from end-hosts. Thus, hot key-value pairs can be retrieved from the network at sub-RTT without reaching end-host compute nodes.

Assisting (Accelerating) Distributed Systems. Replication and consistency are integral components of data center applications to ensure availability. Several works in the past couple of years have looked into novel ways of assisting and accelerating such distributed applications by offloading functionalities to the network data-plane. NetPaxos [30] offloads Paxos [48] algorithm to be run in the network data-plane thus achieving acceleration upto an order of magnitude. NoPaxos [53] demonstrates an overhead-free consistent replication using network ordering. Eris [52] and Harmonia [86] perform in-network concurrency control and in-network conflict detection at network line-rate. NetChain [41] performs in-network sub-RTT coordination services and achieves a magnitude increase (compared to ZooKeeper [1]) in throughput with lower latency.

Researchers have also looked at accelerating distributed applications including layer-4 load balancing (SilkRoad [50]), time synchronization (DPTP [44]), string searching (PPS [39]), data analytics (iPipe [56]), machine learning classification (Xiong et al. [82]) and high-speed trading (Jepsen et al. [40]).

6 Discussion and Conclusion

6.1 Network Monitoring and Debugging

While network monitoring and debugging traditionally have always been complex, programmable networks provide the necessary visibility to look into precise information (timestamps, counters, etc.) needed by network operators. The recent works on network monitoring are based on creating post-cards or appending information on packets (based on custom queries specified by network operator). The information is aggregated at a central controller. These approaches are too expensive to be enabled continuously due to overhead of the disk write space and speed (at controller) and communication to the controller. Hence, the telemetry reporting is enabled only when an operator observes a problem. However, by doing so, they may end up missing the historical information which is critical to debug the root-cause of a network fault. Hence, research is needed in areas to identify faults at runtime and exporting only necessary and precise information needed to debug the root-cause of faults.

6.2 In-Network Computing

The challenges to in-network computing comes from the following aspects of programmable switches : 1) Computation: limited computation could be done at line-rate, 2) Storage: limited SRAM capacity and, 3) Programmability: challenging to express traditional applications in the network data-plane which is based on packets. It is necessary to keep these constraints in mind while deciding the right functions or applications to place in the network.

While the recent years have observed several novel applications being offloaded to the network, there have also been debates on whether it is right for the network to handle such applications and what type of functions and/or applications should be offloaded to the network [61] [68]. One suggested guideline [68] is to weigh the gain against the additional overhead in terms of computation and storage in the network data-plane.

Finally, to exploit the programmability available, it is important to have high-level compiler for network applications, which can dynamically place parts of a network application on either end-host, SmartNIC or programmable switches. While, there have been some recent works such as SNAP [23], Sluice [65] and Chipmunk [31], there is much more work needed in this area to identify and automatically extract the right primitives from end-host applications to run in the network.

References

[1] Apache ZooKeeper. http://zookeeper.apache.org.

[2] Broadcom Trident 3. https://packetpushers.net/broadcom-trident3-programmable-varied-volume/.

[3] Cavium xpliant ethernet switch product family. https://goo.gl/xzfLLo.

[4] Cisco Nexus 34180YC and 3464C Programmable Switches. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-3000-series-switches/datasheet-c78-740836.pdf.

[5] Floodlight OpenFlow Controller. http://www.projectfloodlight.org/floodlight/.

[6] Forwarding and control element separation (forces). https://datatracker.ietf.org/wg/forces/documents/.

[7] General switch management protocol (gsmp). https://datatracker.ietf.org/wg/gsmp/about/.

[8] In-band Network Telemetry. https://p4.org/assets/INT-current-spec.pdf.

[9] Innovium teralynx. https://www.innovium.com/products/teralynx/.

[10] Intel flexpipe. https://goo.gl/PzPudG.

[11] NetFlow. https://en.wikipedia.org/wiki/NetFlow.

[12] Onos project. https://onosproject.org/.

[13] Opendaylight. https://www.opendaylight.org/.

[14] Portable Switch Architecture. https://p4.org/p4-spec/docs/PSA-v1.0.0.pdf.

[15] Pox sdn controller. https://noxrepo.github.io/pox-doc/html/.

[16] Ryu sdn controller. https://osrg.github.io/ryu/.

[17] SDNet Packet Processor. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1012-sdnet-packet-processor.pdf.

[18] Sflow. https://en.wikipedia.org/wiki/SFlow.

[19] A simple network management protocol (snmp). https://www.ietf.org/rfc/rfc1157.txt.

[20] The world's fastest and most programmable networks. https://www.barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/.

[21] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages 1–300, 2008.

[22] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The switchware active network architecture. *IEEE Network*, 1998.

[23] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. SNAP: Stateful Network-Wide Abstractions for Packet Processing. In *SIGCOMM*, 2016.

[24] J. Biswas, A. A. Lazar, J. . Huard, Koonseng Lim, S. Mahjoub, L. . Pau, M. Suzuki, S. Torstensson, Weiguo Wang, and S. Weinstein. The IEEE P1520 standards initiative for programmable network interfaces. In *IEEE Communications Magazine*, 1998.

[25] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 2014.

[26] 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. In *SIGCOMM*, 2013.

[27] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and O. Rottenstreich. Catching the Microburst Culprits with Snappy. In *SelfDN*, 2018.

[28] D. Clark. The design philosophy of the darpa internet protocols. In *SIGCOMM*, 1988.

[29] G. Cormode and M. Muthukrishnan. Approximating Data with the Count-Min Sketch. *IEEE Softw.*, 2012.

[30] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé. NetPaxos: Consensus at Network Speed. SOSR, 2015.

[31] X. Gao, T. Kim, A. K. Varma, A. Sivaraman, and S. Narayana. Auto-generating Fast Packet-Processing Code Using Program Synthesis. In *HotNets*, 2019.

[32] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[33] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. Sonata: Query-driven Streaming Network Telemetry. In *SIGCOMM*, 2018.

[34] 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 *NSDI*, 2014.

[35] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik. Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *SIGCOMM*, 2017.

[36] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever. Blink: Fast Connectivity Recovery Entirely in the Data Plane. In *NSDI*, 2019.

[37] Q. Huang, P. P. C. Lee, and Y. Bao. Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *SIGCOMM*, 2018.

[38] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. In *SIGCOMM*, 2013.

[39] T. Jepsen, D. Alvarez, N. Foster, C. Kim, J. Lee, M. Moshref, and R. Soulé. Fast String Searching on PISA. In *SOSR*, 2019.

[40] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé. Packet Subscriptions for Programmable ASICs. In *HotNets*, 2018.

[41] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. Netchain: Scale-free sub-rtt coordination. NSDI, 2018.

[42] 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. SOSP, 2017.

[43] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *APSys*, 2018.

[44] P. G. Kannan, R. Joshi, and M. C. Chan. Precise time-synchronization in the data-plane using programmable switching asics. In *SOSR*, 2019.

[45] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM*, 2002.

[46] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *SOSR*, 2016.

[47] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. S. , and T. Woo. The SoftRouter Architecture. In *HotNets*, 2004.

[48] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 1998.

[49] L. K. Lazar, A.A. and F. Marconcini. Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture. In *IEEE Journal of Selected Areas in Communications*, 1996.

[50] J. Lee, R. Miao, C. Kim, M. Yu, and H. Zeng. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *SIGCOMM*, 2017.

[51] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *SOSP*, 2017.

[52] J. Li, E. Michael, and D. R. K. Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. SOSP, 2017.

[53] 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 *OSDI*, 2016.

[54] X. Li, R. Sethi, M. Kaminsky, D. G. Andersen, and M. J. Freedman. Be fast, cheap and in control with switchkv. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 31–44, Santa Clara, CA, 2016. USENIX Association.

[55] Y. Li, R. Miao, C. Kim, and M. Yu. Flowradar: A better netflow for data centers. In *NSDI*, 2016.

[56] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta. Offloading Distributed Applications Onto smartNICs Using iPipe. In *SIGCOMM*, 2019.

[57] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya. Incbricks: Toward in-network computation with an in-network cache. ASPLOS, 2017.

[58] X. Liu, M. Shirazipour, M. Yu, and Y. Zhang. Mozart: Temporal coordination of measurement. In *SOSR*, 2016.

[59] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica. Distcache: Provable load balancing for large-scale storage systems with distributed caching. In *FAST*, 2019.

[60] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *SIGCOMM*, 2016.

[61] J. McCauley, A. Panda, A. Krishnamurthy, and S. Shenker. Thoughts on load distribution and the role of programmable switches. *SIGCOMM Comput. Commun. Rev.*, 2019.

[62] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation

in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[63] E. Z. N. Feamster, J. Rexford. The Road to SDN: An Intellectual History of Programmable Networks. In *SIGCOMM Comput. Commun. Rev.*, 2014.

[64] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-Directed Hardware Design for Network Performance Monitoring. In *SIGCOMM*, 2017.

[65] V. Natesh, P. G. Kannan, A. Sivaraman, and R. Netravali. Sluice: Network-Wide Data Plane Programming. In *SIGCOMM Posters and Demos*, 2019.

[66] N.Kang, Z.Liu, J.Rexford, and D.Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *CoNEXT*, 2013.

[67] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. In *IEEE Communications Surveys Tutorials*, 2014.

[68] D. R. K. Ports and J. Nelson. When should the network be the computer? In *HotOS*, 2019.

[69] T. Qu et al. SQR: In-network packet loss recovery from link failures for high-reliability datacenter networks. In *ICNP*, 2019.

[70] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis. In-network computation is a dumb idea whose time has come. In *HotNets*, 2017.

[71] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik. Scaling distributed machine learning with in-network aggregation. *CoRR*, abs/1903.06701, 2019.

[72] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira. Elmo: Source Routed Multicast for Public Clouds. In *SIGCOMM*, 2019.

[73] N. K. Sharma, A. Kaufmann, T. Anderson, C. Kim, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the Power of Flexible Packet Processing for Network Resource Allocation. In *NSDI*, 2017.

[74] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. Approximating fair queueing on reconfigurable switches. In *NSDI*, 2018.

[75] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-Hitter Detection Entirely in the Data Plane. In *SOSR*, 2017.

[76] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith. Turboflow: Information Rich Flow Record Generation on Commodity Switches. In *EuroSys*, 2018.

[77] 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 *ATC*, 2018.

[78] C. . Tai, J. Zhu, and N. Dukkipati. Making large scale deployment of rcp practical for real networks. In *INFOCOM*, 2008.

[79] P. Tammana, R. Agarwal, and M. Lee. Distributed network monitoring and debugging with switchpointer. In *NSDI*, 2018.

[80] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. Ants: a toolkit for building and dynamically deploying network protocols. In *IEEE Open Architectures and Network Programming*, 1998.

[81] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. Ofrewind: Enabling record and replay troubleshooting for networks. In *ATC*, 2011.

[82] Z. Xiong and N. Zilberman. Do Switches Dream of Machine Learning?: Toward In-Network Classification. In *HotNets*, 2019.

[83] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In *SIGCOMM*, 2018.

[84] N. Yaseen, J. Sonchack, and V. Liu. Synchronized network snapshots. In *SIGCOMM*, 2018.

[85] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *NSDI*, 2013.

[86] H. Zhu, Z. Bai, J. Li, E. Michael, D. R. K. Ports, I. Stoica, and X. Jin. Harmonia: Near-Linear Scalability for Replicated Storage with In-Network Conflict Detection. In *VLDB*, 2020.