A Vision for Runtime Programmable Networks

Jiarong Xing
Rice University

Yiming Qiu
Rice University

Kuo-Feng Hsu
Rice University

Hongyi Liu
Rice University

Matty Kadosh
Nvidia

Alan Lo
Nvidia

Aditya Akella
UT Austin

Thomas Anderson
University of Washington

Arvind Krishnamurthy
University of Washington

T. S. Eugene Ng
Rice University

Ang Chen
Rice University

Abstract

Our community has made significant progress in developing programmable network infrastructure, starting from the control plane and expanding to the data plane. As a latest trend, network devices are becoming *runtime* programmable while serving live traffic. This allows for reprogramming of individual device programs at fine-grained timescales to add or remove network functions. Many applications and services, however, need control over a combination of devices, including end host stacks, NICs, and switches, to accomplish their goals. We lay out our vision for *runtime programmable networks*, building upon device-level features to provide live, network-wide, runtime reprogramming. A whole-stack approach is needed with new programming models, compiler support, and network management abstractions. We outline a research agenda as a call to arms to the community.

CCS Concepts

• **Networks** → **Programmable networks**;

Keywords

Programmable networks

ACM Reference Format:
Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Hongyi Liu, Matty Kadosh, Alan Lo, Aditya Akella, Thomas Anderson, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. 2021. A Vision for Runtime Programmable Networks. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3484266.3487377

1 Introduction

Our community has made significant progress in making the network infrastructure programmable. Network programmability started with the control plane, but has rapidly expanded to the data plane. Programmable data plane devices, such as switches [5, 7, 51], NICs [2, 8], FPGAs [4, 61], and software targets [48], have gained popularity. This in turn has changed the way in which we control and operate our networks. In a programmable network, operators are capable of customizing the network infrastructure end-to-end, by writing and deploying network programs at the host stacks, NICs, or switches. Without any need for hardware upgrades, new functions can be introduced and unused ones removed by reflashing the devices with different programs.

Up until recently, one missing piece of this puzzle has been *runtime programmability*. Existing work [16, 24, 38, 40, 42, 67] has extensively studied the opportunities afforded by compile-time programmability—i.e., customizing device behaviors by compiling a new network program and reflashing the data plane before the device starts to serve traffic. In compile-time programmable networks, devices that need to be "repurposed" are first isolated by management operations (e.g., draining traffic), reconfigured with a different program, before they are redeployed to the network again. In contrast, *runtime programming* of network devices, while keeping the network disruption-free, enables a new paradigm. With runtime programmable devices, reprogramming takes place at much finer timescales hitlessly. The data plane is kept live while program changes are reconfigured. Example runtime programmable switches include Nvidia/Mellanox Spectrum series (with P4) [66], Broadcom Trident4 and Jericho2 (with NPL). Runtime programmable targets also includes FPGAs and software switches, as they are inherently capable of live, partial reconfiguration.

This paper investigates a vision that we call FlexNet, which leverages the trend of device-level runtime programmability, but considers broader design principles for an end-to-end *runtime programmable network*.

FlexNet envisions a network infrastructure that shapeshifts in response to real-time change. At any point, the end-to-end network is optimally tuned for the current requirements and traffic workloads. But if network requirements change in the next minute, reconfigurations across devices will present the network as a new infrastructure. This requires runtime programming of individual devices as a building block, but also synchronized reconfigurations across the network. Network functions migrate seamlessly from one location to another,

HotNets'21, November 10-12, 2021, Virtual Event, United Kingdom

ACM ISBN 978-1-4503-9087-3/21/11…$15.00
https://doi.org/10.1145/3484266.3487377

both vertically (host stacks vs. NICs. vs. switches) and horizontally (end-to-end network paths). They run atop devices with different architectures, programming models, and performance characteristics. Security defenses are summoned to the network via precise injection to attack locations for real-time mitigation, and they dynamically scale in and out based on attack traffic volume. End-to-end, the network is piloted by a central controller that maintains a global view of the topology and traffic patterns, as well as the locations and resource requirements of the network apps. The software controller initiates centralized management operations, but they are handed over to data plane hardware when feasible for efficient, distributed execution.

While ambitious, we believe this goal is within reach. The emergence of runtime programmable data planes points to the technological feasibility. The need for runtime programmability is evidenced by industry support from disparate vendors [3, 5, 6, 10] and academic research that approximates this capability [18, 30, 62, 70]. Though not yet pervasive across all devices, commercial incentives for adding such support in upcoming device models seem compelling. In fact, we view runtime programmability as a crucial step to the overall success of programmable networks. It simultaneously addresses two critical needs in large networks: rapid innovation of network features [23] and their deployment with high network availability and near-zero downtime [28].

1.1 A Case for Runtime Programmable Networks

FlexNet enables a range of powerful use cases.

Dynamic apps. Programmable networks have found many applications [29, 34, 38, 40, 50] but today's apps are statically compiled into the network and cannot change at runtime. Recent projects call out this limitation and propose approximating solutions. They essentially work by baking all needed logic at compile time but changing how it is used from the control plane. DynamiQ [18] designs a monitoring system where query operators are flexibly mapped at runtime to compile-time allocated resources. Mantis [70] hardcodes all runtime response logic at compile time, and invokes different responses at runtime by modifying control registers. HyPer4 [30] emulates different network programs with a virtualization layer. In contrast, runtime programmable networks offer direct support for dynamic program changes. One does not need to anticipate all network requirements in advance or statically bake everything into the network.

Real-time security. Statically-baked network programs are particularly problematic for security defenses [37, 42, 67–69], as attacks are in nature fast-changing and difficult to anticipate or provision for. Runtime programmable defenses can be summoned into the network on-the-fly and retired when attacks subside. Such defenses are also elastic, capable of scaling, replicating, and migrating to other locations based on changing attack strengths and patterns. Real-time defense deployments also enable hot-patching the network against zero-day attacks before a permanent fix is rolled out.

Live infrastructure customization. Whole-network infrastructure customization is challenging in today's datacenters. Deploying new transport protocols [39, 43], for instance, requires changes not only to host kernels but also telemetry and congestion control (CC) algorithms at the NICs and switches. The optimal choice of CC algorithms further depends on the mix of applications and workloads [49], which fluctuate dynamically at runtime. FlexNet enables quick, incremental upgrades of the end-to-end infrastructure at runtime.

Tenant extensions. Cloud datacenters must accommodate the varied networking requirements of each tenant. The number of virtual networks and their needs change rapidly due to tenant churn. FlexNet allows tenants to inject customer-specific network extensions (e.g., new CC algorithms at the hosts and NICs, or custom security defenses at the switches) as they arrive. Tenant departures trigger program removal to trim the network and release unused resources.

1.2 Research Challenges

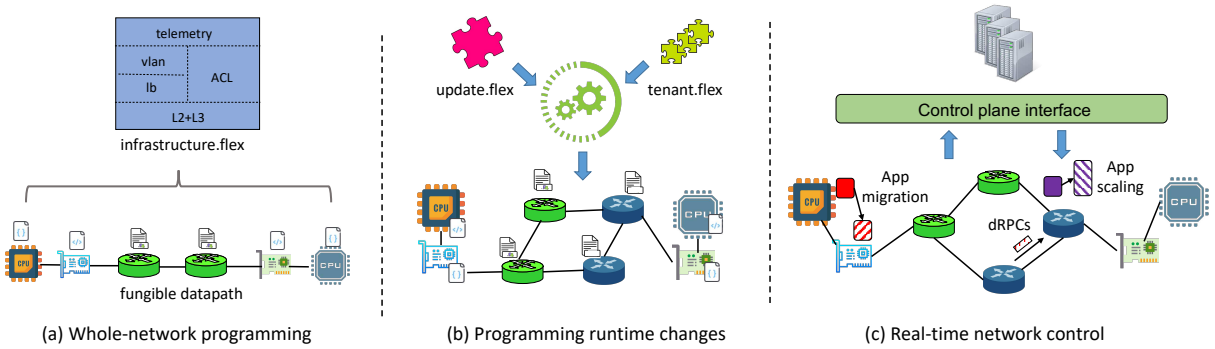
Realizing this vision requires a whole-stack approach that rethinks how FlexNet networks should be programmed, optimized, and managed at scale.

Whole-network programming. Building upon device-level runtime programmability as a basis, FlexNet aims to enable whole-network runtime programming end-to-end. This raises interesting research questions on developing suitable programming abstractions for vertical and horizontal function distribution, which simultaneously take into account device heterogeneity in terms of programming models, architectures and performance characteristics.

Programming runtime changes. Specifying runtime changes is different from writing a new program from scratch. It requires incremental programming support. FlexNet permits runtime modifications to the "infrastructure" program (e.g., as supplied by the network operator), as well as "extension" programs (e.g., provided by the tenants) in a modular manner. Runtime changes are programmed in an incremental manner to avoid intrusive modifications to the base program.

Compiling fungible programs. FlexNet also introduces research opportunities for new network compiler designs. Existing compilers [27, 36] assume a non-fungible network, and their primary concern is to bin-pack program elements into resource-constrained devices. Runtime programmable networks enable a new operating point for compilers because network resources become fungible. When they are not in use, programs are removable to release resources. Therefore, FlexNet compilers have the option of optimizing for alternative goals (e.g., performance, energy) even if they come with resource overheads. Extra resources can be reallocated or reclaimed elsewhere in the network.

Compiling runtime changes. Compiling changes into the network must be done in a least-intrusive manner to avoid significant resource reallocation and shuffling across the network. Redistribution of program elements may also require recompilation to a different target, as well as conversion and migration of program state to a different representation.



(a) Whole-network programming (b) Programming runtime changes (c) Real-time network control

Figure 1: The FlexNet vision and its key components. FlexNet provides a "fungible datapath" abstraction and enables runtime whole-network programming. The compiler analyzes the program and runtime extensions, and distributes the components vertically and horizontally. The network is piloted by a central controller for real-time management.

Piloting runtime programmable networks. Significantly greater responsibility will fall unto the network management system in a runtime programmable network. Traditional management and control platforms [33, 35, 55] are not up to the task: they manage devices that perform tasks of the same nature (i.e., forwarding), so their primary job is to optimize the forwarding behavior (e.g., alleviate congestion via traffic engineering). But as FlexNet devices host different apps, which in turn consume variable amounts of resources, exhibit different performance characteristics, and are capable of runtime migration, their management becomes more challenging. New control plane API operating at the "app" level is needed for management. Control operations may also be handed over to the data plane for efficient execution. Network control needs to be aware of mixed deployments with compile-time programmable and non-programmable elements. Consensus, availability, and fault tolerance also need to be revisited for developing logically centralized but physically distributed controllers [13, 45].

In the rest of this paper, we outline the FlexNet vision and a research agenda.

2 Runtime Programmability

Our vision is based upon the trend that network devices are becoming *runtime* programmable. We describe the current ecosystem of runtime programmable targets and their varying degrees of flexibility.

Switches. Switch vendors are increasingly exposing runtime programmability in their ASICs. Our recent work has developed runtime programming support for Nvidia/Mellanox Spectrum series of switches, reconfigurable in P4 [66]. While keeping the device live, match/action tables can be added and removed on-the-fly without packet loss. Parser states can be similarly manipulated to add and remove header types and protocols. Program changes complete within a second, and during this transition, packets are either processed by the new program or old one in a consistent manner. Broadcom Trident4 and Jericho2 switches are runtime programmable in NPL. Dynamic tables can be runtime reconfigured to perform a different task or change to different table keys without downtime.

FPGAs and SmartNICs. FPGAs and SoC-based SmartNICs are inherently more flexible than switching ASICs, both for compile-time and runtime reconfiguration. Live, partial reconfiguration of FPGAs has been extensively studied in the hardware community [60]. Traditional FPGA development requires Verilog or VHDL programming, but when FPGAs are used as network devices, high-level languages like P4 have gained wide support. SoC-based SmartNICs, like Netronome Agilio, Nvdia/Mellanox BlueField, and Pensando DPUs, enclose general-purpose SoC cores and are programmable in C. As of late, they also ship P4 compilers for the NICs. As SoC cores are general-purpose in nature, no fundamental barrier exists in supporting other languages (e.g., NPL). For FPGA and SoC targets, their raw capability of runtime reconfiguration carries over to the network programs that they host. For instance, when hosting a P4 program on such a device, partial reconfiguration primitives for tables and parsers would similarly apply. On these targets, runtime programming primarily requires more mature tooling support that specifically target network programs in P4 or NPL.

Host kernels. The kernel network stack allows for runtime customization via the eBPF framework [12]. eBPF kernel extensions are constrained C programs, and can be injected to the network stack without any disruption. Runtime reconfigurations occur at eBPF program level, e.g., by reloading a different program.

To summarize, runtime programmable variants exist for all classes of popular targets for network programming. Although their programming models, flexibility, and performance characteristics vary, the existing ecosystem already presents sufficient building blocks to develop *end-to-end runtime programmable networks*. We believe the time is ripe to explore this FlexNet vision.

3 Open Problems

An array of research challenges exists in the development of runtime programmable networks. Figure 1 illustrates.

Scenario. In the ensuing discussion, we assume a generic deployment scenario where the network infrastructure is operated by its owner but individual tenants dynamically arrive and depart. The network provider maintains an "infrastructure" program, which implements basic functions for the network as well as utility functions for management and control. Tenants provide "extension" programs that are dynamically injected into and removed from the network. The infrastructure program forms a trusted base, and the extensions are admitted by the network owner after access control validation. Extension programs are isolated from each other and from the infrastructure code via, e.g., VLAN-based isolation mechanisms. Tenant arrivals trigger the generation of new VLAN configurations from the control plane, as well as infrastructure program changes to accommodate the new extensions. Departures achieve opposite effects. All programs are continuously updated in real time, and changes are integrated into the network seamlessly without downtime.

3.1 Runtime Whole-Network Programming

From a whole-stack perspective, runtime network programming goes much beyond programming packet processing pipelines, e.g., in P4 [7], NPL [5], or their combination [27]. Vertically, host kernel stacks and SmartNICs expose general-purpose programming models in C or restricted C (e.g., eBPF). They are also capable of a wider range of network customization tasks, e.g., custom congestion control [17] and transport protocols [15, 31, 43], or TCP offloads [44]. While these domain-specific tasks are also constrained in nature, they are very different from packet-oriented processing. To enable whole-network customization vertically and horizontally, new programming models and abstractions are required.

Abstractions. We envision that a FlexNet program is written against a network abstraction that hides away the details of vertical and horizontal distribution, as well as device heterogeneity in terms of architectures, performance characteristics, and programming models. The compiler analyzes the program, and automatically splits it to the physical network. Existing abstractions like the "one big switch" model [16] serve similar goals for networking programming, but are insufficient for capturing vertical implementations across host stacks, NICs, and switches. They also do not model a network infrastructure where resources can be reallocated, reclaimed, and redistributed at runtime.

We call this abstraction a "fungible datapath", which logically models a whole-stack network device, including L2 and L3 functionalities, but also programmable transport protocols [15, 43] or even higher-layer offloads [40]. Under the hood, it is implemented on a physical slice of the end-to-end network. The compiler analyzes the datapath program and determines which components should run where. Within a fungible datapath, program components may freely migrate and elastically scale in and out on different physical devices. The shape and size of the physical slice are additionally regulated by the network control policies and the negotiated SLAs. Tenant datapaths are laid atop the infrastructure datapath with proper access control isolation.

Programming languages. We envision that fungible datapaths require a domain-specific language that mixes match/-action-style packet processing and eBPF-style offloads, which we will call FlexBPF. In our vision, FlexBPF should expose a logical and constrained form of network state, organized in key/value "maps". The FlexBPF control flow and operations need to go well beyond matches and actions, so as to fully leverage device programmability. With constrained state, FlexBPF programs are analyzable to certify bounded execution, well-behavedness, and to enable automated compilation to constrained targets [72]. FlexBPF programs express programmable congestion control, transport protocols, constrained higher-layer offloads, and packet-processing pipelines in the fungible datapath.

The logical key/value maps maintain a virtualized view of network state at different layers. Virtualizing network state is crucial, as individual devices have drastically different ways of implementing this state. Consider some examples. The P4 language standard defines stateful *registers and counters* as "extern" constructs that are up to the device vendor. PoF devices expose a different abstraction: *flow state instruction sets* [51]. Nvidia/Mellanox devices pursue yet another route: *stateful tables* that are indexed with flow key, with flow insertions and removals performed in the data plane [58]. If a program assumes a specific way of state encoding (e.g., registers), function migration becomes difficult. In FlexBPF, the compiler selects the proper state encodings for different program components based on the target devices. Program migration carries its state in this logical representation.

3.2 Programming Runtime Changes

Specifying runtime changes, whether updating the infrastructure program or injecting tenant extensions, presents a different set of challenges. Runtime changes require *incremental* programming and compilation support to minimize intrusiveness.

Incremental upgrades. Updates to the infrastructure or tenant programs, by themselves, need not specify a complete network processing stack. They are simply additions, deletions, or changes to the existing programs. Our goal is to develop a domain-specific language that concisely specify where, when, and how an existing FlexNet program is updated. Programs in this DSL precisely model the changes that need to be made, without having to re-specify the entire stacks all over again. For instance, this DSL may expose name matching utilities (e.g., via pattern matches on match/action tables and actions) to programmatically select and modify the firewall- or CC-related functions in the base program. The FlexNet compiler jointly analyzes the pattern matching program with the base program and regenerates program changes exactly where needed.

Datapath composition. Runtime changes also include injecting or removing an end-to-end tenant datapath. For these situations, FlexNet needs to enable datapath composition.

Recent work [52] has developed modular, composable abstractions for P4 programs (e.g., one modular for L2 processing and another for L3). Similar properties are useful for FlexBPF programs, but additional analyses are required—e.g., the tenant extensions have restricted access control permissions; different tenants may inject logically-sharable code that present optimization opportunities or conflicting datapaths that need to be resolved.

3.3 Compiling Fungible Programs

Runtime programmable networks open up new operating regimes for compiler design. Existing network compilers [27, 36] assume that device resource limits are an unyielding constraint and primarily focus on bin-packing programs within available resources. However, since a runtime programmable network can dynamically remove unused functions, device resources become fungible. This enables a new search space for compiler optimizations. For instance, the FlexNet compiler may operate in multiple iterations. If compiling a FlexNet datapath to its resource slice fails, the compiler recursively invokes optimization primitives for its datapath to perform resource reallocation and garbage collection, before attempting another round of compilation.

Resource fungibility. Resource fungibility varies across device architectures, and shuffling program elements may also result in a physical datapath with different performance characteristics.

(i) RMT. The RMT (reconfigurable match table) architecture [19] adopts a pipeline model with a fixed number of stages, and packets are processed by MA (match/action) tables stage by stage. Example switch ASICs that adopt this architecture include Intel FlexPipe and Tofino. For Tofino, resources in the same hardware stage are fungible. By adding runtime support to reconfigure individual stages in a live manner, tables can be potentially shuffled across stages and all pipeline resources would become fungible.

(ii) dRMT. The disaggregated RMT architecture [22], on the other hand, removes the static stage boundaries by disaggregating compute from memory. A set of MA processors execute a P4 program in a run-to-completion manner for each incoming packet. MA table entries are physically separated from the processors in SRAM or TCAM. Unrestricted by stage boundaries, any processor can access any table, at any point in the P4 program. Our previous work also builds upon a similar architecture as implemented in the Nvidia/Mellanox Spectrum model [66]. On this architecture, memory and action resources are fungible due to disaggregation.

(iii) Tiles, Elastic Pipes. Tiled and Elastic Pipe architectures, as adopted by Broadcom's Trident4 and Jericho2, are yet another class. For Trident4, hash and index tiles are realized in SRAM; alongside TCAM tiles, they are exposed to the programmer [10]. NPL programs determine inter-tile connectivity and tile programmability. Jericho2's Elastic Pipe architecture, on the other hand, consists of a standard pipeline of stages that is extended by a Programmable Elements Matrix (PEM) [3]. On these architectures, fungibility occurs within the same tile types and the PEM elements.

(iv) SmartNICs, FPGAs, and Hosts. Resources are essentially fully fungible on these architectures.

Across the network, resources that lie on the same network path are fungible as traffic flow through a sequence of devices [27]. By co-desiging routing and placement mechanisms for a logical datapath, more opportunities will open up (e.g., via routing detours to a program component).

Performance and energy optimizations. Leveraging resource fungibility, the FlexNet compiler is able to explore additional objectives beyond resource bin-packing. Resources on switching ASIC, SmartNICs, FPGAs, and hosts, though fungible, have different performance characteristics. Therefore, our compiler must take performance SLA into consideration when it maps a logical datapath to the physical infrastructure. In a similar spirit, different targets also have varied energy consumption envelopes [57]. By leveraging this fungibility layer, FlexNet is able to shuffle resource around and optimize for the current workload regarding network energy consumption. Moreover, fungible resources also allow for optimizations that trade performance/energy goals with resource utilizations. Merging two match/action tables, for instance, will lead to increased memory usage due to a table "cross product", but it saves one table lookup time and reduces latency for packet processing on certain architectures.

Incremental recompilation. When compiling runtime changes into the network, FlexNet also needs to perform incremental recompilation. FlexNet not only needs to generate optimized programs, but also needs to minimize the amount of resource reshuffling by identifying "maximally adjacent reconfigurations" that lead to non-intrusive redistribution. As resource shuffling may also affect datapath performance, FlexNet needs to re-certifying SLA objectives as well. A fine balance between compilation time and optimization levels is necessary. For fast reactions to network changes, it may be desirable to generate non-optimal implementations in a shorter turnaround time.

3.4 Real-time Network Control

New network control and management systems are required to effectively pilot runtime programmable networks. Existing designs, such as OpenFlow SDN controllers [33, 55] and traditional network management systems [20, 21], are primarily concerned with managing *forwarding* behaviors. In these traditional networks, devices perform tasks of the same natur—i.e., routing and forwarding—despite "micro-level" device heterogeneity (e.g., differences in hardware vendors, generations, or control interfaces). Therefore, network control primarily performs traffic engineering to alleviate congestion [33] and carries out disruption-free network updates [55]. A runtime programmable network, however, requires a very different kind of network piloting, as "macro-level" heterogeneity exists across devices that host different apps. Deciding on optimal app locations, reasoning about app resource requirements, elastic app scaling, resilient state replication, app

migration, as well as the traditional goal of managing routing behaviors, are all up to the network management system.

Control plane abstractions. In FlexNet, we propose to expose abstractions for app-level network management. The P4Runtime standard [11] has a set of control plane API to manage and interact with P4-capable devices, but they operate at the data plane element level, e.g., manipulating counters, meters, and table rules. This is a starting point, but FlexNet also requires higher-level abstractions to manage the apps. For instance, the controller is able to "name" in-network apps by their URIs (instead of, say, IP addresses), and perform management operations using the URI as a handle (e.g., expand a certain resource type). In other words, application-centric abstractions are needed as first-class primitives. Their translation into lower-level commands (e.g., via P4Runtime) is done automatically by the FlexNet management system.

Data plane execution. We envision that the network control operations are invoked by the control plane, but their execution may take place partially or entirely in the data plane. Unlike existing network control platforms [63] that manage software-based entities, FlexNet controller needs to efficiently manipulate in-network, data plane programs. These apps process and produce linespeed data, and their internal state also mutates per-packet at nanosecond timescales. If all control operations are performed in software, many tasks become extremely challenging or infeasible.

Consider migrating a stateful network app (e.g., one that maintains a count-min sketch). As the sketch state is updated for each packet, copying state via control plane software is impossible [41]. Recent work has developed tools to perform state migration entirely in data plane [41, 65]. FlexNet needs more control primitives of this kind that are realizable in hardware data planes. In particular, in-network monitoring, execution tracking, and diagnosis primitives will prove useful for runtime programmable app management, as such networks experience higher dynamics. These "utility" functions for network control do not have a persistent footprint inside the network, but are injected in real-time for maintenance tasks and removed soon after. In mixed deployments of runtime programmable, compile-time programmable, and non-programmable devices, FlexNet also needs to account for the topological locations of these network elements.

dRPCs. Realizing control operations in the data plane also requires handling devices with different programming capabilities and performance characteristics. Since not every device will support all operations, we envision that the infrastructure program will provide a set of data plane RPC services for common utilities (e.g., app migration or state replication). Tenant datapaths need not reinvent the wheel but rather invoke these remote services via data plane RPC calls (dRPCs). Tenant programs may also expose tenant-specific RPC services that the infrastructure program can invoke. Service discovery occurs either at control plane or via an in-network RPC registry and discovery protocol in real time.

Fault tolerance and consistency. To detect and tolerate device failures, the FlexNet controller replicates important network state in a logical datapath across multiple physical devices. State consistency is ensured via state replication and update protocols [71]. Functional updates to a logical datapath also need application-level, consistent packet processing, which goes beyond controlling the order of rule updates [46], and varied levels of consistency guarantees may apply [66]. It is the network controller's job to ensure that traffic in the datapath is routed through the correct sequence of network devices to receive processing. For large networks, logically centralized controllers are realized in physically distributed nodes, which brings classic distributed systems concerns on consensus and availability [13, 45]. In a runtime programmable network, developing a new consistency model and enforcing it across distributed controller nodes also raises interesting research questions.

4 Related Work

Network programming. Recent work has developed support for network programming both for single devices [5, 7] and distributed environments [16, 27, 54]. Pronto [26] lays out a vision for closed-loop network programming, observation, and verification. FlexNet is closely related to these work, but it investigates *runtime* network programmability.

Runtime reconfigurability. Runtime reprogrammability has been studied in several contexts. The architecture community has extensively explored the capability of live, partial reconfiguration of FPGAs [60]. The OS community has considers kernel reconfiguration via eBPF [12, 59] and live VM migration [9, 53]. The networking community has developed support for eBPF-style reconfiguration in XDP [32] and SmartNIC offloads [1]. FlexNet explores the vision of *whole-network* reconfigurability at runtime.

Active networks. Active networking research has laid the foundation for many important developments in network programmability [14, 47, 56]. Recent work also revisits this line of work and its progression to programmable networks today [25, 64]. FlexNet is aligned with this vision and investigates the next step in making programmable networks even more flexible than they are today.

5 A Call to Arms

Networking research is entering a "golden era". Ossification concerns start to dissipate and exciting possibilities are opening up. Making the network infrastructure end-to-end programmable at *runtime* is not only technologically feasible today, but also pays great dividends. We believe that runtime programmability is an attractive next step in our community's intellectual trajectory. Joint community efforts from both academia and industry are needed in this endeavor.

Acknowledgments: We thank the anonymous reviewers for their helpful feedback on this work. This work was partially supported by NSF grants CNS-1565277, CNS-1717039, CNS-1801884, CNS-1856636, CNS-1942219, and CNS-2105868.

References

[1] Agilio CX SmartNICs. https://www.netronome.com/products/agilio-cx/.

[2] BlueField SmartNIC Ethernet. https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet.

[3] Jericho2. https://www.broadcom.com/products/ethernet-connectivity/switching/stratadnx/bcm88850.

[4] Mellanox Innova-2 Flex Open Programmable SmartNIC. https://www.mellanox.com/products/smartnics/innova-2-flex/.

[5] nplang. https://github.com/nplang.

[6] Nvidia/Mellanox Spectrum Ethernet Switches. https://www.nvidia.com/en-us/networking/ethernet-switching/spectrum-sn4000/.

[7] The P4 language repositories. https://github.com/p4lang.

[8] SmartNIC Overview - Netronome. https://www.netronome.com/products/smartnic/overview/.

[9] Supporting live migration of vms communicating with bare-metal rdma endpoints. https://www.openfabrics.org/wp-content/uploads/2021-workshop-presentations/402_Hansen_PVRDMA.pdf.

[10] Trident4 boosts enterprise switch capacity to 12.8 terabit. http://www.gazettabyte.com/home/2019/7/11/trident-4-boosts-enterprise-switch-capacity-to-128-terabit.html.

[11] P4Runtime. https://p4.org/p4-runtime/.

[12] What is eBPF? https://ebpf.io/.

[13] A. Akella and A. Krishnamurthy. A highly available software defined fabric. In *Proc. HotNets*, 2014.

[14] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. j Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare active network architecture. *IEEE Network*, 12(3):29–36, 1998.

[15] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal near-optimal datacenter transport. In *Proc. SIGCOMM*, 2013.

[16] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. SNAP: Stateful network-wide abstractions for packet processing. In *Proc. SIGCOMM*, 2016.

[17] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff. Enabling programmable transport protocols in high-speed NICs. In *Proc. NSDI*, 2020.

[18] R. Bhatia, A. Gupta, R. Harrison, D. Lokshtanov, and W. Willinger. DynamiQ: Planning for dynamics in network streaming analytics systems. *arXiv preprint arXiv:2106.05420*, 2021.

[19] 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. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[20] X. Chen, Y. Mao, Z. M. Mao, and K. van der Merwe. Declarative configuration management for complex and dynamic networks. In *Proc. CoNEXT*, 2010.

[21] X. Chen, Z. M. Mao, and J. V. der Merwe. PACMAN: a platform for automated and controlled network operations and configuration management. In *Proc. CoNEXT*, 2009.

[22] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, et al. drmt: Disaggregated programmable switching. In *Proc. SIGCOMM*, 2017.

[23] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E. C. Zermeno, E. Rubow, J. A. Docauer, et al. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *Proc. NSDI*, 2018.

[24] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé. NetPaxos: Consensus at network speed. In *Proc. SOSR*, 2015.

[25] N. Feamster, J. Rexford, and E. Zegura. The road to SDN: An intellectual history of programmable networks. *ACM SIGCOMM CCR*, 44(2):87–98, 2014.

[26] N. Foster, N. McKeown, J. Rexford, G. Parulkar, L. Peterson, and O. Sunay. Using deep programmability to put network owners in control. *SIGCOMM Comput. Commun. Rev.*, 50(4):82–88, 2020.

[27] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu. Lyra: A cross-platform language and compiler for data plane programming on heterogeneous ASICs. In *Proc. SIGCOMM*, 2020.

[28] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. Evolve or die: High-availability design principles drawn from Google's network infrastructure. In *Proc. SIGCOMM*, 2016.

[29] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. Sonata: Query-driven streaming network telemetry. In *Proc. SIGCOMM*, 2018.

[30] D. Hancock and J. van der Merwe. HyPer4: Using P4 to virtualize the programmable data plane. In *Proc. CoNEXT*, 2016.

[31] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wöjcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proc. SIGCOMM*, 2017.

[32] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller. The express data path: Fast programmable packet processing in the operating system kernel. In *Proc. CoNEXT*.

[33] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. SIGCOMM*, 2013.

[34] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, P. Tammana, and D. Walker. Contra: A programmable system for performance-aware routing. In *Proc. NSDI*, 2020.

[35] 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 *Proc. SIGCOMM*, 2013.

[36] L. Jose, L. Yan, G. Varghese, and N. McKeown. Compiling packet programs to reconfigurable switches. In *Proc. NSDI*, 2015.

[37] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo. Programmable in-network security for context-aware BYOD policies. In *Proc. USENIX Security*, 2020.

[38] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *Proc. SOSR*, 2016.

[39] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu. Hpcc: High precision congestion control. In *Proc. SIGCOMM*, 2019.

[40] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya. IncBricks: Toward in-network computation with an in-network cache. In *Proc. ASPLOS*, 2017.

[41] S. Luo, H. Yu, and L. Vanbever. Swing State: Consistent updates for stateful and programmable data planes. In *Proc. SOSR*, 2017.

[42] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev. NetHide: Secure and practical network topology obfuscation. In *Proc. USENIX Security*, 2018.

[43] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proc. SIGCOMM*, 2018.

[44] Y. Moon, S. Lee, M. A. Jamshed, and K. Park. AccelTCP: Accelerating network applications with stateful TCP offloading. In *Proc. NSDI*, 2020.

[45] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker. SCL: Simplifying distributed SDN control planes. In *Proc. NSDI*, 2017.

[46] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent updates for software-defined networks: Change you can believe in! In *Proc. HotNets*, 2011.

[47] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart packets for active networks. In *Proc. OpenArch*, 1999.

[48] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford. PISCES: A programmable, protocol-independent software switch. In *Proc. SIGCOMM*, 2016.

[49] A. Singhvi, A. Akella, D. Gibson, T. F. Wenisch, M. Wong-Chan, S. Clark, M. M. K. Martin, M. McLaren, P. Chandra, R. Cauble, H. M. G. Wassel, B. Montazeri, S. L. Sabato, J. Scherpelz, and A. Vahdat. 1RMA:

Re-envisioning remote memory access for multi-tenant datacenters. In *Proc. SIGCOMM*, 2020.

[50] 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 *Proc. USENIX ATC*, 2018.

[51] H. Song. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In *Proc. HotSDN*. ACM, 2013.

[52] H. Soni, M. Rifai, P. Kumar, R. Doenges, and N. Foster. Composing dataplane programs with μp4. In *Proc. SIGCOMM*, 2020.

[53] R. Stoyanov and M. J. Kollingbaum. Efficient live migration of linux containers. In *International Conference on High Performance Computing*. Springer, 2018.

[54] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. Dehon, and B. T. Loo. Flightplan: Dataplane disaggregation and placement for P4 programs. In *Proc. NSDI*, 2021.

[55] P. Sun, A. Arefin, R. Mahajan, J. Rexford, L. Yuan, and M. Zhang. A network-state management service. In *Proc. SIGCOMM*, 2014.

[56] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *ACM SIGCOMM CCR*, 26(2):5–18, 1996.

[57] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman. The case for in-network computing on demand. In *Proc. EuroSys*, 2019.

[58] A. Tulumello, M. Bonola, S. Pontarelli, M. Kadosh, and Y. Piasetzki. Extending P4 to Realize a Scalable Flow Caching Mechanism. https://opennetworking.org/wp-content/uploads/2021/05/2021-P4-WS-Angelo-Tulumello-Slides.pdf, 2021.

[59] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)*, 53(1):1–36, 2020.

[60] K. Vipin and S. A. Fahmy. FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications. *ACM Comput. Surv.*, 51(4), 2018.

[61] 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 *Proc. SOSR*, 2017.

[62] T. Wang, X. Yang, G. Antichi, A. Sivaraman, and A. Panda. Isolation mechanisms for high-speed packet-processing pipelines. *arXiv preprint arXiv:2101.12691*, 2021.

[63] Y. Wang, E. Keller, B. Biskeborn, J. Van Der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. *ACM SIGCOMM Computer Communication Review*, 38(4):231–242, 2008.

[64] D. Wetherall and D. Tennenhouse. Retrospective on "towards an active network architecture". *SIGCOMM Comput. Commun. Rev.*, 49(5):86–89, Nov. 2019.

[65] J. Xing, A. Chen, and T. E. Ng. Secure state migration in the data plane. In *Proc. SIGCOMM SPIN*, 2020.

[66] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen. Runtime programmable switches. In *Proc. NSDI (to appear)*, 2022.

[67] J. Xing, Q. Kang, and A. Chen. Netwarden: Mitigating network covert channels while preserving performance. In *Proc. USENIX Security*, 2020.

[68] J. Xing, W. Wu, and A. Chen. Architecting programmable data plane defenses into the network with fastflex. In *Proc. HotNets*, 2019.

[69] J. Xing, W. Wu, and A. Chen. Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries. In *Proc. USENIX Security*, 2021.

[70] L. Yu, J. Sonchack, and V. Liu. Mantis: Reactive programmable switches. In *Proc. SIGCOMM*, 2020.

[71] L. Zeno, D. Ports, J. Nelson, and M. Silberstein. SwiShmem: Distributed shared state abstractions for programmable switches. In *Proc. HotNets*, 2020.

[72] K. Zhang, D. Zhuo, and A. Krishnamurthy. Gallium: Automated software middlebox offloading to programmable switches. In *Proc. SIGCOMM*, 2020.