The Slow Path Needs an Accelerator Too!

Annus Zulfiqar, Ben Pfaff[†], William Tu[†], Gianni Antichi[‡*], Muhammad Shahbaz
Purdue University [†]*VMware* [‡]*Queen Mary University of London* [*]*Politecnico di Milano*

ABSTRACT

Packet-processing data planes have been continuously enhanced in performance over the last few years to the point that, nowadays, they are increasingly implemented in hardware (i.e., in SmartNICs and programmable switches). However, little attention is given to the slow path residing between the data plane and the control plane, as it is not typically considered performance-critical.

In this paper, we show that the slow path is set to become a new key bottleneck in Software-Defined Networks (SDNs). This is due to the growth in physical network bandwidth (200 Gbps is becoming common in data centers) and topological complexity (e.g., virtual switches now span hundreds of physical machines). We present our vision of a new Domain Specific Accelerator (DSA) for the slow path at the end host that sits between the hardware-offloaded data plane and the logically-centralized control plane. We discuss open problems in this domain and call on the networking community to creatively address this emerging issue.

CCS CONCEPTS

• **Networks** → **Programmable networks**; *In-network processing*; • **Hardware** → **Networking hardware**.

KEYWORDS

Slow path; Megaflow cache; SDN; P4; OVS; DSA

1 INTRODUCTION

When we think of Software-Defined Networking (SDN), we typically think of a data plane managed by a logically centralized control plane. This separation is pervasive throughout our modern computing landscape, in areas ranging from data centers [4, 42, 43, 64, 67], to wide-area [28], to 4G/5G mobile core [48], to service meshes [12], and more. The control plane performs computationally taxing decision-making on a per-flow basis (using a general-purpose CPU), and the data plane caches the outcome of the decision as flow rules and applies it to every packet (using a dedicated ASIC), Figure 1(a).

This separation of the control plane and data plane is how SDN works in theory. However, in almost all real settings, a third component links the control plane and the data plane, an entity known as the *slow path*, as shown in Figure 1(b). For example, the slow path is the switch OS [15] in hardware switches (e.g., Intel Tofino [24, 25]), the userspace logic in virtual switches (e.g., OVS [46, 52]), the PGW/SMF in 4G/5G networks [48], and an infrastructure layer in service meshes (e.g., Istio [26]). These slow paths are not just responsible for populating data-plane caches (e.g., flow tables) with updates from the control plane, but also perform myriads of other critical tasks for the correct and timely operation of the environment they are running in [13, 52, 58]. For example, without the OS, a hardware switch would fail to update its tables quickly in response to changing network conditions (e.g., link failures and microbursts) [17, 75]. Similarly, the absence of the userspace logic in virtual switches [52] would lead to excessive load on the control plane, which would have to handle flow setup for each new flow. The same holds true for 4G/5G networks and service meshes (§2).

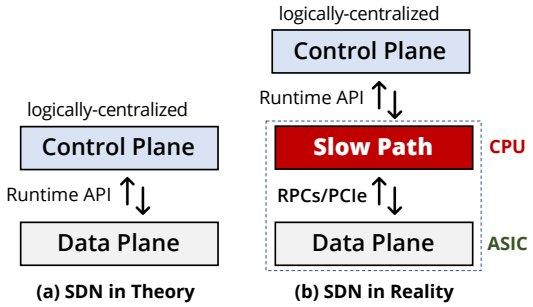

Figure 1: Slow path is the backbone of SDN systems.

This paper is about these slow paths[1] running on a CPU. We want to bring the attention of the community to this largely ignored component of the networking stack, which is on the brink of becoming a bottleneck in emerging, next-generation networks. So far, as a networking community, our focus has been on accelerating the data plane using dedicated silicon chips (e.g., Tofino for hardware switches [24, 25] or FPGA-based SmartNICs for virtual switches [22, 39, 40, 71]). Similar efforts are underway in accelerating the user plane of the 4G/5G mobile core [33, 47] and the data plane (i.e., sidecar proxy [56]) of service meshes using programmable ASICs [27]. This trend is likely to continue as link rates rise beyond 200 Gbps, both at end-hosts and in the core of the network. Increasing link rates are not just straining the data plane but also the slow path. Likewise, the evolving compute and application landscapes (i.e., mega-scale, multi-tenant clouds and highly disaggregated micro-services) are further stressing the slow path when scaling to an ever-increasing number of tenants and services [59].

Corresponding author: Annus Zulfiqar (zulfiqaa@purdue.edu)

[1] A slow path is operating at a much faster timescale than a control plane—the "slow" in slow path is in comparison to the data plane, similar to what slow start is in TCP [49].

SDN Area	Offloaded Data Plane	Slow Path Example(s)
Virtual Switches	NIC [41, 73]	Userspace `ovs-vswitchd` [45]
Hardware Switches	RMT [6] Trident [8]	Switch OS `Stratum` [15], `SONiC` [76]
Service Meshes	P4 hardware [27]	Sidecar Proxy `Istio` [26]
5G Mobile Core	P4 hardware [33, 47]	Session Manager `AMF`, `SMF` [48]

Table 1: Many areas accelerate their data plane with dedicated hardware. Yet, the slow path still resides on general-purpose CPUs, facing myriad scalability and performance overheads.

We argue and show that state-of-the-art CPU-based platforms alone, whether on the end host, SmartNIC, or switch, are a poor fit for the slow path (§2). Instead, like the data plane, *the slow path needs an accelerator too.* In this paper, we make the case for such an accelerator for the end-host slow path using Open vSwitch (OVS) as an exemplar (§3).

In OVS, the slow path serves two purposes: (1) to handle infrastructure protocols (such as BFD, LACP, IGMP, and OpenFlow) and flow-table updates that a data-plane ASIC (e.g., a switch or NIC) otherwise cannot handle, as a kind of exception handler, and (2) to abstract away the resource limitations of the data-plane ASIC, providing the control plane with an abstraction of any-size flow table while installing only the most valuable subset in the data plane. Ideally, these infrastructure protocols and table updates should all run in the data plane; however, today's data-plane accelerators are optimized for forwarding user traffic only, using match-action table (MAT) pipelines [6, 36]. They treat these protocols and updates as exceptions, which are handled by a (CPU-based) slow path [36]. Hence, the slow path handles a *subset* of the data-plane traffic that requires *exceptional* processing. These *exceptions* include the sub-linerate processing that requires complex control-flow, compute, or memory resources, which the data plane is not designed to handle. These slow-path processing requirements are difficult (or impossible) to express using MATs, so the slow path today is implemented on a CPU, as shown in Table 1.

At first glance, this division of labor may seem like the right approach, that, clearly, high-volume user traffic should be handled in the data-plane ASIC, and slow-path traffic (i.e., infrastructure protocols and table updates) be processed on a CPU. However, as we show in §3, slow-path traffic has grown commensurately with user traffic over the years. Allocating more CPU resources directly reflects as higher operational expenses (OPEX), a cause of primary concern for the network operators running today's mega-scale cloud data centers because of per-core pricing [68]. That's why operators of these data centers aim at provisioning a single CPU core for their virtual switches [66], making the rest available for revenue-generating tenant workloads. Offloading the slow path to recent ARM-based SmartNICs at the end-hosts is also not feasible: (1) they would not scale to ever-increasing link rates (exceeding 200 Gbps) [29], and (2) their maximum power usage is orders of magnitude higher than an equivalent ASIC-based NIC [2, 18, 19, 72], further raising OPEX. The situation is similar for hardware switches: recent work [75] shows that the onboard CPUs cause tens or hundreds of microseconds of delay when reacting to network conditions, compared to the ASIC that is operating in nanoseconds. In short, for slow paths to scale in efficiency and performance in the next-generation networks, we must move away from purely CPU-based platforms to accelerated ones, similar to what we have been doing for the data plane (i.e., with switching ASICs) [6].

To make our case, we begin by detailing the widespread presence of slow paths in the modern computing ecosystem (§2.1), and recent efforts to scale it using system-level enhancements (§2.2). Next, we study Open vSwitch (§3), a production-quality virtual switch, and thoroughly evaluate its slow path (§4).

2 BACKGROUND & MOTIVATION

Despite its practical significance, the slow path has been a neglected area of networking research in the 14-year history of SDN [35]. In this section, we provide a background and understanding of how pervasive this slow path is (§2.1) and what challenges it is (and will be) facing in the emerging computing landscape (§2.2).

2.1 The Slow Path is Everywhere

Like control and data planes, a slow path is present in almost all areas of SDN, ranging from traditional virtual and hardware switches to, more recently, 5G mobile core and service meshes, as summarized in Table 1.

Slow Path in Virtual Switches. Popular and widely used virtual switches, including OVS [52, 66], Microsoft VFP [13], and Google Snap [34], use a slow path to separate (and couple) the control and data plane. Over time, the per-packet (data-plane) processing functions have been offloaded to dedicated hardware ASICs (e.g., OVS offload [16, 32, 41, 73], Accelnet for VFP [14], 1RMA for Snap [60]). The slow path, on the other hand, has remained on general-purpose CPUs, where it is responsible for the creation and deletion of flows, maintaining large flow caches to abstract away resource limits of data-plane ASICs, executing control protocols (e.g., for link failure detection, multicast management, and link bonding and rebalancing, described in more detail in §3), and operations like packet sampling.

Slow Path in Hardware Switches. Programmable switches include both an ASIC (e.g., Tofino [24, 25] and Broadcom Trident [7, 8]) and a CPU. The former processes packets and the latter runs a switch OS, such as Stratum [15] or SONiC [76]. The switch OS on the CPU is the slow path, serving as a bridge between the central controller (e.g., ONOS [42]) and the ASIC. It deals with flow installation, exception handling (e.g., control protocols), and other requests from the control plane (e.g., register or counter reads). Recent research also uses the slow path (i.e., switch OS) for more compute-intensive purposes. Mantis [75], for example, uses the CPU to react to congestion events in microseconds, while ACC [74] uses the switch CPU to perform reinforcement learning using a DNN to autotune ECN marking thresholds in data planes.

Slow Path in 5G Mobile Core. The mobile core also exhibits a data-plane (called the *user-plane*) and control-plane separation. Starting with 5G, the cellular industry is adopting a service-based architecture, by splitting its monolithic processing stack into components (e.g., AMF, SMF, and UPF) that run as independent services [48]. The UPF is the data-plane component of the core, responsible for routing and forwarding packets. The SMF and other components form the slow path, responsible for adding flows in the UPF and handling various protocol-specific signaling (e.g., attach, detach) [48]. The industry is moving toward offloading the UPF to dedicated programmable ASICs [33, 37, 47], but little attention has been given to the slow-path components [5]. As 5G networks need to connect trillions of mobile and IoT devices to the Internet edge [38], the load on its slow path is even higher, thus demanding better slow-path implementations.

Slow Path in Service Meshes. Service meshes [56] enable inter-service communication to scale beyond the microservice architecture. A set of network proxies, called *side cars*, are deployed alongside application code within the service node. These proxies comprise the data plane, while a controller (e.g., Istio [26]) orchestrates inter-service communication at the application layer, i.e., the slow path. Here again, the division between the data plane and the slow path is well-defined and, while researchers have been looking into accelerating the proxies [27], not much has been done yet on the other side. The service mesh's slow path is feature-rich, implementing functions such as timeouts, retries, circuit-breaking, service gateways, service discovery, declarative traffic management, and adaptive load-balancing [3, 12, 26]. Benchmarks indicate the slow path contributes up to 44% of service-mesh CPU utilization, indicating a CPU-based platform to be a performance bottleneck [31].

2.2 Challenges in Scaling the Slow Path

It is common to dedicate CPU cores to run the infrastructure code (e.g., the slow path and hypervisor) and to use dedicated hardware to offload the data plane [44]. The new generation of SmartNICs (e.g., Intel IPU [23], Nvidia Bluefield DPU [40], Xilinx Alveo SN-1000 [71], and AMD Pensando [1]), can offload infrastructure code to on-NIC ARM cores, which frees host CPU cycles for applications. However, ARM-based SmartNICs do consume a lot more power than ASIC-based NICs: a SmartNIC without CPUs typically consumes 4.6 W [18], whereas one with onboard ARM cores consumes between 55 and 75 W [2, 19, 72]. This increases the operating costs for cloud vendors. Secondly, offloading infrastructure processing to NICs does not address performance bottlenecks inherent to CPUs (e.g., variable processing latencies, non-deterministic response time, HoL blocking).

In conversations with a major cloud provider, we learned that the large volumes of traffic entering the virtual switch cause control packets to experience excessive head-of-line (HoL) blocking, when contending with other slow-path traffic (e.g., flow misses). As tenant traffic (including control packets) is typically tunneled, the NIC's data plane cannot prioritize these packets, without requiring deep-packet inspection. Proposals to offload individual control protocol to the data plane are also being explored, both as dedicated hard-wired logic [36, 70] or programmable ASICs (e.g., P4-based Tofino [6]). Hardwiring these protocols as fixed logic makes them harder to change or evolve later, whereas current P4-based data-plane programmable models are too restrictive to express stateful operations. Our insight is that to sustain performance and efficiency while retaining flexibility and feature velocity, we need a domain-specific architecture for the slow path, just like we have RMT [6] for the data plane.

3 CASE STUDY: OPEN VSWITCH (OVS)

To study the challenges associated with running a slow path on CPUs, we picked Open vSwitch (OVS) [52]—a widely used open-source virtual switch inside cloud data centers—as our case study. OVS consists of a slow path and a fast path. Initially, the fast path used to run as software in the Linux kernel [51] or userspace [44, 66]. However, today it is offloaded to the NIC's data plane, as a flow-table classifier, to handle the rising link rates (200 Gbps) entering the datacenter end-hosts [57]. Popular NIC vendors now support OVS fast-path offload on their data-plane ASICs (e.g., Nvidia ConnectX-6 Dx [41], Intel IPU SmartNIC [32], and Xilinx Alveo SN1000 SmartNIC [73]). Yet, the slow path still runs on CPUs, either host CPUs or SmartNIC's onboard CPUs [16].

3.1 The OVS Slow Path

The OVS slow path performs three main tasks, shown in Figure 2: executing OpenFlow pipelines on new incoming flows; updating caches, including the hardware flow tables; and processing control packets pertaining to the infrastructure.

a) Flow Caches and Tables. When a packet belonging to a new flow arrives at the NIC for the first time, it misses

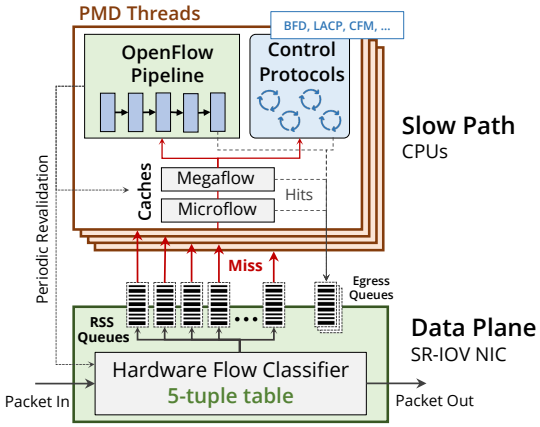

Figure 2: The design of OVS: data plane offloaded to an SR-IOV NIC, and slow path running on server CPUs.

in the hardware flow table. The NIC enqueues the packet in one of the RSS queues, picked using the hash of the packet's 5-tuple. Each of these queues is connected to a "poll-mode driver" (PMD) thread, over PCIe. To sustain high throughput, the OVS slow path runs multiple PMD threads, each with its own set of caches, control processes, and OpenFlow pipeline.

Upon entering the slow path, the packet again misses in two layers of flow caches: microflow and megaflow. The microflow cache is optimized to handle high-bandwidth "elephant" flows, and the megaflow cache holds more general-purpose entries that can match wildcard rules. Finally, the packet enters either the OpenFlow pipeline if it is a cache miss, or the control-protocol handler if it is a control packet.

b) The OpenFlow Pipeline. The pipeline implements a sequence of OpenFlow-compliant flow tables [65] that network operators can configure from the centralized control plane. At runtime, the control plane specifies the flow rules (i.e., the match and action instructions) and other properties of the pipeline using the OpenFlow and OVSDB APIs [50, 65, 69]. OVS passes each missed packet through the OpenFlow pipeline table by table, composing each table's actions into a final set of actions, plus a set of caching instructions. The slow path executes these instructions on these packets and, if the instructions permit, installs a rule in the micro-/mega-flow cache of its PMD thread and the hardware flow table in the NIC. Later packets for the same flow match these rules. The slow path also periodically revalidates the caches and the hardware flow table, to ensure correctness against up-to-date rules in the OpenFlow pipeline.

c) Control Protocols. The OVS slow path needs to process various infrastructure (control) protocols as well, such as those for link-failure detection (CFM [21] and BFD [30]), link bonding and rebalancing (LACP [20]), and multicast group management (IGMP) [10]. It also runs protocols for packet sampling (e.g., sFlow and IPFIX) for sending packets to the remote collector or the control plane.

3.2 Slow Path Bottlenecks

Traffic Patterns and Emerging Trends. Bottlenecks arise when slow-path traffic, such as cache misses and control packets, arrives faster than it can be processed. Cache misses in the data-plane ASIC are a common source of bottlenecks that has received attention both in industry [52] and research [54, 55]. Most often, these misses are because of the NIC's data plane's limited memory and feature support (i.e., it can only do 5-tuple matches), which causes flows matching on other fields to be processed by the slow path. The amount of incoming traffic a slow path can handle is also proportional to the number of PMD threads required (1 thread per core); as the traffic rate increases, the load on these threads also increases. With sufficient load, congestion in these threads causes HoL blocking and queuing delays, which in turn increases flow setup time. We show in §4 that increasing the number of PMD threads does not help.

Changing traffic patterns can also cause high miss rates. For example, some traffic patterns (such as microflows with varying source and destination addresses or packets for P2P rendezvous applications) can cause excessive misses [52]. Similarly, configuring rules to construct a complicated OpenFlow pipeline in the slow path, without following OVS best practices, can render the megaflow cache ineffective, so that each entry essentially ends up doing an exact match (leading to more misses). Bad actors can also exploit the flow tables remotely, by taking advantage of OVS classifier weaknesses to send pathological traffic that causes cache explosion [11].

Periodic Cache Revalidation. The slow path periodically revalidates the data plane cache, usually every second, to ensure correctness against up-to-date OpenFlow rules in the slow path. OVS revalidates in dedicated slow-path threads, which visit all the data-plane cache entries. For each entry, they simulate a traversal of the OpenFlow pipeline and compare the output to what is currently installed in the cache, and update the entry if necessary. The slow-path revalidation threads, therefore, must be able to verify the whole cache in under one second. And, if the cache entries exceed a prescribed limit, then further cache-flow entries cannot be installed—resulting in all packets of new arriving flows consistently entering the slow-path (as demonstrated in §4), which hampers performance due to an overloaded slow path.

Infrastructure Packet-processing Overhead. The packet-processing load of control protocols can also produce bottlenecks. The worst case happens for the BFD and CFM protocols when monitoring the liveness of peers. When slow-path processing falls behind, e.g., due to HoL blocking under excessive load (e.g., cache misses), it can prevent BFD and CFM packets from being processed and replied to in a timely fashion. This can cause a host's peers to conclude that the link is down and that they should choose an alternate host, which

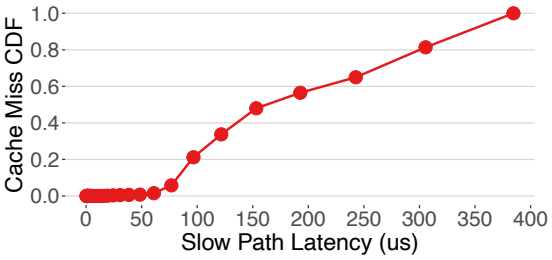

Figure 3: Empirical distribution of cache miss processing times. We only show a quarter of the total number of cache misses that are processed in under 400 μs. The rest take more than 400 μs (not shown here).

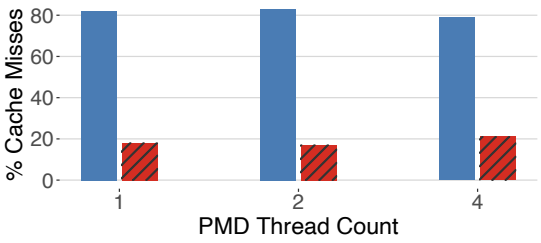

Figure 4: Number of served (blue) and dropped (red) cache misses vs. number of PMD threads. The number of PMD and revalidator threads are kept equal.

increases the traffic to that host and potentially causes further disruption. Using longer protocol timeouts eases the issue, but it can potentially slow down the detection of genuine network or host failures.

4 IMPLICATIONS OF THE SLOW PATH

To understand the efficiency and performance implications of the slow path, we set up a testbed with two machines equipped with dual-port ConnectX-6 DX SmartNICs [39] connected back-to-back. Both machines have two Intel Xeon Platinum 8358P processors with 64 cores, running at 2.60 GHz with 512 GB of memory. They both run Ubuntu 22.04 with kernel version 5.15.0-33-generic. We also set up OVS 2.17.90 with DPDK 22.03.0-rc1 and Mellanox OFED 5.6-1.0.3 drivers (`mlx5_core`), and configured SR-IOV on the NIC with OVS offload. One machine ran the TRex traffic generator and a custom DPDK script to generate a ClassBench-rules-compatible CAIDA [9] traffic profile, on-the fly [55, 63], to the other machine, which is the device under test (DUT).

Cache misses cause highly skewed slow-path processing times with long tails. We installed 200K rules in OVS, and tested its cycles per cache-miss performance against real-world CAIDA [9] traffic profile taken from the Equinix datacenter in January 2019. A custom DPDK script [55] took CAIDA's traffic profile (e.g., flow locality, inter-arrival times) and generated compatible packets on-the-fly by replacing the 5-tuples from the traffic with 5-tuples from ClassBench rules to maintain the properties of a realistic traffic trace. We configured OVS to use 4 RXQs for the physical interface with 4 PMD threads, and generated the traffic at 10 Gbps to send a total of 790 million packets. With these rules, we observed a total of 893K cache misses in OVS, 1.5% of which were dropped at all four PMD threads. Figure 3 shows a cumulative distribution of the CPU cycles consumed by the cache misses in the OVS slow path. Worth noting is the two orders of magnitude variation (between median and tail) in CPU processing times for these cache misses[2] (all matching on 5 tuples). A cache miss can take anywhere from 30K to millions of clock cycles in slow-path processing. These processing times will be even higher in a realistic environment, with OpenFlow rules matching other header fields (31 in a representative NSX deployment [44]), not just a 5-tuple. These rules can further burden the slow-path, leading to even higher tail latencies. The impact on applications' performance can be significant: Memcached suffers by over 50% performance hit in the presence of 50 μs added delay [53].

Increasing flow rules saturate caches and increase the revalidation cost. We installed 450K ClassBench rules (utilizing ACL seed-5 profile) and used OVS internal counters to measure the number of cache misses when OVS processes CAIDA traffic (with rule-compatible headers) at 25 Mpps. We sent traffic for 12 seconds and repeated it with different PMD thread counts (while having equal or more RXQs configured on the ports). The number of revalidator threads are also kept equal to the number of PMD threads. The number of successfully processed cache misses (that resulted in a cached entry) versus the dropped (did not result in a cached entry) are shown in Figure 4. Cache misses are dropped due to the cost of revalidation (§3), which increases with larger cache sizes. To keep the cost within limits, additional cache misses are dropped, which causes all later packets of those flows to enter the slow path. The consequence of this bottleneck is wasted CPU cycles in processing all the subsequent *non-cachable* upcalls, which could have been avoided if the revalidation cost were affordable.

CPU-based slow paths struggle to keep up with increasing traffic volume. These are two further consequences of the results shown in Figures 3 and 4. First, the volume of slow-path traffic (cached or dropped) on a single PMD core is about 8K requests per second when running at a link rate of 25 Mpps. With link rates projected to exceed 200 Gbps (288 Mpps for minimum-sized packets), the slow-path traffic would rise to roughly 92K requests per second, which on a state-of-the-art server CPU, running at 2.6 GHz translates to 28K clock cycles (10.8 μs) per request. Figure 3 indicates that, even with simple rule matching, there is an order of magnitude of variation in completion times for different slow-path

[2]Figure 3 shows only 25% of the actual upcalls. The OVS performance counters aggregate all upcalls consuming over 1M clock cycles. These upcalls end up on the right side of this CDF (not shown).

requests, making a single PMD core dedicated to handle slow-path traffic grossly insufficient, hence calling for CPU scaling just to handle the slow-path traffic.

Second, scaling the number of cores only partially addresses the problem. As shown in Figure 4, as we increase the PMD thread count, the slow-path traffic volume does not reduce since cache misses occur when new flows arrive regardless of having more cache space (§3). The revalidation cost of flow caches demands limited cache sizes and increasing the revalidation threads doesn't benefit. Moreover, the dropped traffic volume also increases and the number of clock cycles left for slow-path processing per request per CPU at higher rates does not benefit much.

Uncertainties in control-packet processing yields devastating datacenter-wide consequences. OVS runs the BFD protocol for link failure and fault detection. Typical production environments configure pairwise BFD among virtual tunnel endpoints. If a BFD packet is delayed while waiting behind long slow-path queues (Figure 3), OVS might falsely label a link down and, as a consequence, stop forwarding traffic to a node. Similarly, OVS handles IGMP snooping in the slow path to detect multicast routers. Delayed processing of IGMP packets would cause multicast packets to be incorrectly delivered to obsolete ports, increasing traffic on those links and exacerbating slow-path traffic volume. Creating dedicated queues for processing such packets is not feasible since these packets are tunneled in real environments (i.e., encapsulated in a VxLAN/Geneve tunnel header), and most hardware NICs cannot classify and prioritize using inner protocol headers [1, 23, 40, 71].

5 DISCUSSION & CALL TO ARMS

Towards a Slow-Path Accelerator & Beyond. Slow-path bottlenecks (§4) can be removed by offloading to a dedicated accelerator optimized for slow-path processing. Recently, Intel has proposed Infrastructure Processing Units (IPUs) [23] for offloading the entire infrastructure code (including the hypervisor) from host machines to NICs. These processors and other similar efforts [71] provide on-NIC ARM cores and FPGAs to offload the control plane. While this helps in freeing the host machine CPU cores, the problems with tail latencies and scaling core count for increasing traffic (e.g., for ARM NICs), as well as the expertise needed to program them (e.g., for FPGA NICs) still remain. Considering these factors, we argue that the best way forward is to build a domain-specific architecture (DSA) specifically designed to offload the slow path that sits in between the (ASIC-based) data plane and the centralized control plane.

The main challenge we envision is finding the right set of abstractions that will make our architecture generic enough to support many use cases (e.g., from virtual switching to service meshes). Analogous to what RMT [6] represents for

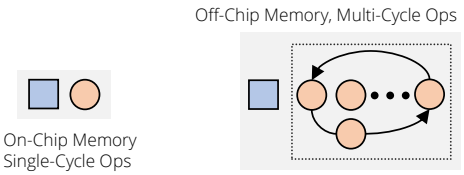

Figure 5: The match-compute abstraction can process multi-cycle DAGs with off-chip memories (e.g., HBMs).

packet forwarding, Taurus [62] for machine learning, and PIFO [61] for scheduling, we need to identify common primitives in the slow path for infrastructure control protocols and other slow-path operations—abstracting away their implementation details and providing a programmable target specialized for the slow path. We believe the DSA for slow path needs three properties: *Predictable response times:* It must guarantee bounded tail latency given the trend towards deterministic response times in feed-forward networking architectures. *Fast table updates:* It should provide fast and high-bandwidth IO processing for flow caching, as it is a critical function of a slow path. *Large memory pools:* It must support large pools of memory (using HBMs) to augment the limited on-chip memory of the data planes (§3).

From Match-Action to Match-Compute. Match-action pipelines in today's RMT switches implement simple (single-cycle) operations to ensure per-packet processing at multi-Tbps speeds. The slow path, on the other hand, operates at lower speeds (e.g., 40–200Gbps) and, therefore, can perform more complicated (multi-cycle) operations per packet per stage. To program these slow-path accelerators, we could extend the match-action abstraction (Figure 5a) to a more general *match-compute* abstraction (Figure 5b), where a match is followed by a DAG of compute primitives. This ability to expend more cycles per stage can enable many interesting capabilities necessary to support slow-path applications. For example: *(a) Stateful and event-driven processing:* For fast reactivity, we need event-based primitives and timers to perform stateful operations, to support flexible pipelines and packet-processing capabilities. *(b) Iterative processing:* To perform periodic operations, we require loops with deep pipelines and parallelism, to iteratively traverse data structures (e.g., for cache revalidation in OVS).

We believe that building a DSA for the slow path is an essential next step in our community's intellectual trajectory. We, therefore, call on academic and industry researchers and developers to join us on this mission.

Acknowledgements. This research was supported by NSF award CNS 2211381; by ACE, one of the seven centers in JUMP 2.0, an SRC program sponsored by DARPA; and by the European Union's Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001, "RESTART").

REFERENCES

[1] AMD. last accessed: 11/28/2022. Pensando. https://www.amd.com/en/accelerators/pensando.

[2] AMD. last accessed: 11/28/2022. Pensando DSC-200 Distributed Services Card. https://www.amd.com/system/files/documents/pensando-dsc-200-product-brief.pdf.

[3] Gianni Antichi and Gábor Rétvári. 2020. Full-Stack SDN: The Next Big Challenge?. In *SOSR*.

[4] Antrea last accessed: 11/28/2022. Antrea: Enhance pod networking and enforce network policies for Kubernetes clusters. https://antrea.io/.

[5] Abhik Bose, Shailendra Kirtikar, Shivaji Chirumamilla, Rinku Shah, and Mythili Vutukuru. 2022. AccelUPF: Accelerating the 5G User Plane Using Programmable Hardware. In *SOSR*.

[6] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *ACM SIGCOMM*.

[7] BROADCOM. last accessed: 11/28/2022. BCM88690. https://www.broadcom.com/products/ethernet-connectivity/switching/stratadnx/bcm88690.

[8] BROADCOM. last accessed: 11/28/2022. Trident4 / BCM56880 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series.

[9] CAIDA. last accessed: 11/30/2022. The CAIDA UCSD anonymized internet traces. https://www.caida.org/catalog/datasets/passive_dataset/.

[10] Bradley Cain, Dr. Steve E. Deering, Bill Fenner, Isidor Kouvelas, and Ajit Thyagarajan. last accessed: 11/30/2022. Internet Group Management Protocol, Version 3. https://www.rfc-editor.org/info/rfc3376.

[11] Levente Csikor, Dinil Mon Divakaran, Min Suk Kang, Attila Kőrösi, Balázs Sonkoly, Dávid Haja, Dimitrios P. Pezaros, Stefan Schmid, and Gábor Rétvári. 2019. Tuple Space Explosion: A Denial-of-Service Attack against a Software Packet Classifier. In *ACM CoNEXT*.

[12] Alexis de Talhouët. last accessed: 11/30/2022. The evolution of SDN: What service mesh offers telco. https://www.redhat.com/en/blog/evolution-sdn-what-service-mesh-offers-telco.

[13] Daniel Firestone. 2017. VFP: A Virtual Switch Platform for Host SDN in the Public Cloud. In *USENIX NSDI*.

[14] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *USENIX NSDI*.

[15] Open Networking Foundation. last accessed: 11/28/2022. Stratum - Enabling the Era of Next Generation SDN. https://opennetworking.org/stratum/.

[16] Malvika Gupta. last accessed: 11/30/2022. Open vSwitch Offload by SmartNICs on Arm. https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/open-vswitch-offload-by-smartnics-on-arm.

[17] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. 2019. Blink: Fast Connectivity Recovery Entirely in the Data Plane. In *USENIX NSDI*.

[18] Serve The Home. last accessed: 11/28/2022. Intel X710 OCP NIC 3.0 Power Consumption Specs. https://www.servethehome.com/intel-x710-da2-ocp-nic-3-0-review-10gbe-for-the-form-factor/intel-x710-ocp-nic-3-0-power-consumption-specs/.

[19] Serve The Home. last accessed: 11/28/2022. Pensando Distributed Services Architecture SmartNIC. https://www.servethehome.com/pensando-distributed-services-architecture-smartnic/.

[20] IEEE. last accessed: 11/30/2022. IEEE Standard for Information Technology - Local and Metropolitan Area Networks - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications-Aggregation of Multiple Link Segments. https://standards.ieee.org/ieee/802.3ad/1088/.

[21] IEEE. last accessed: 11/30/2022. IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management. http://standards.ieee.org/getieee802/download/802.1ag-2007.pdf.

[22] Intel. last accessed: 11/30/2022. Intel Ethernet Controller 700 Series - Open vSwitch Hardware Acceleration Application Note. https://builders.intel.com/docs/networkbuilders/intel-ethernet-controller-700-series-open-vswitch-hardware-acceleration-application-note.pdf.

[23] Intel. last accessed: 11/30/2022. Intel Infrastructure Processing Units (IPUs) and Smart-NICs. https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html.

[24] Intel. last accessed: 11/30/2022. Tofino: P4-programmable Ethernet switch ASIC that delivers better performance at lower power. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[25] Intel. last accessed: 11/30/2022. Tofino2: Second-generation P4-programmable Ethernet Switch ASIC that Continues to Deliver Programmability without Compromise. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[26] Istio. last accessed: 11/30/2022. Simplify observability, traffic management, security, and policy with the leading service mesh. https://istio.io/.

[27] Anjali Singhai Jain, Mrittika Ganguli, Valas Valancius, and Nupur Jain. last accessed: 11/30/2022. Service Mesh P4 Data Plane. https://opennetworking.org/2022-p4-workshop-gated/.

[28] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined WAN. In *ACM SIGCOMM*.

[29] Georgios Katsikas, Tom Barbette, Marco Chiesa, Dejan Kostic, and Gerald Maguire. 2021. What You Need to Know About (Smart) Network Interface Cards. In *International Conference on Passive and Active Network Measurement (PAM)*.

[30] Dave Katz and David Ward. last accessed: 11/30/2022. Bidirectional Forwarding Detection (BFD). https://www.rfc-editor.org/info/rfc5880.

[31] Kinvolk.io. last accessed: 11/30/2022. Performance Benchmarks Analysis of Istio and Linkerd. https://kinvolk.io/blog/2019/05/performance-benchmark-analysis-of-istio-and-linkerd.

[32] Patricia Kummrow. last accessed: 11/30/2022. The IPU: A New, Strategic Resource for Cloud Service Providers. https://community.intel.com/t5/Blogs/Tech-Innovation/Data-Center/The-IPU-A-New-Strategic-Resource-for-Cloud-Service-Providers/post/1335081.

[33] Robert MacDavid, Carmelo Cascone, Pingping Lin, Badhrinath Padmanabhan, Ajay ThakuR, Larry Peterson, Jennifer Rexford, and Oguz Sunay. 2021. A P4-Based 5G User Plane Function. In *SOSR*.

[34] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: A Microkernel Approach to Host

Networking. In *SOSP*.
[35] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM CCR* (2008).
[36] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. 2018. Hardware-Accelerated Network Control Planes. In *HotNets*.
[37] Napatech. last accessed: 11/30/2022. UPF offload for Napatech Programmable SmartNICs. https://www.napatech.com/support/resources/data-sheets/5g-user-plane-function-upf-offload/.
[38] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. 2021. A Variegated Look at 5G in the Wild: Performance, Power, and QoE Implications. In *ACM SIGCOMM*.
[39] Nvidia. last accessed: 11/30/2022. CONNECTX-6 DX. https://www.nvidia.com/en-us/networking/ethernet/connectx-6-dx/.
[40] Nvidia. last accessed: 11/30/2022. NVIDIA BLUEFIELD DATA PROCESSING UNITS. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.
[41] NVIDIA. last accessed: 11/30/2022. OVS Offload Using ASAP2 Direct. https://docs.nvidia.com/networking/display/MLNXENv531001/OVS+Offload+Using+ASAP2+Direct.
[42] ONOS last accessed: 11/30/2022. ONOS: Open Network Operating System. https://opennetworking.org/onos/.
[43] Open Daylight last accessed: 11/30/2022. Open Daylight: modular open platform for customizing and automating networks of any size and scale. https://www.opendaylight.org/.
[44] Open-vSwitch. last accessed: 11/30/2022. Open vSwitch with DPDK. https://docs.openvswitch.org/en/latest/intro/install/dpdk/.
[45] Open-vSwitch. last accessed: 11/30/2022. ovs-vswitchd(8). https://www.openvswitch.org/support/dist-docs-2.5/ovs-vswitchd.8.txt.
[46] Github Open-vSwitch. last accessed: 11/30/2022. ofproto-dpif-upcall.c. https://github.com/openvswitch/ovs/blob/master/ofproto/ofproto-dpif-upcall.c.
[47] Francesco Paolucci, Davide Scano, Filippo Cugini, Andrea Sgambelluri, Luca Valcarenghi, Carlo Cavazzoni, Giuseppe Ferraris, and Piero Castoldi. 2021. User Plane Function Offloading in P4 switches for enhanced 5G Mobile Edge Computing. In *International Conference on the Design of Reliable Communication Networks (DRCN)*.
[48] Larry L. Peterson, Carmelo Cascone, Brian O'Connor, Thomas Vachuska, and Bruce Davie. 2021. *Software-Defined Networks: A Systems Approach*. Systems Approach LLC.
[49] Larry L. Peterson and Bruce S. Davie. 2021. *Computer Networks: A Systems Approach* (9th ed.). Morgan Kaufmann.
[50] Ben Pfaff and Bruce Davie. last accessed: 11/30/2022. The Open vSwitch Database Management Protocol. https://www.rfc-editor.org/info/rfc7047.
[51] Ben Pfaff and Jesse Gross. last accessed: 11/30/2022. Open vSwitch datapath developer documentation. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/openvswitch.rst.
[52] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *USENIX NSDI*.
[53] Diana Andreea Popescu. last accessed: 11/30/2022. Latency-driven performance in data centers. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-937.pdf.
[54] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2020. A Computational Approach to Packet Classification. In *ACM SIGCOMM*.
[55] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2022. Scaling Open vSwitch with a Computational Cache. In *USENIX NSDI*.
[56] Redhat. last accessed: 11/30/2022. What's a service mesh? https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh.
[57] Gerald Rogers and Pravin Shelar. last accessed: 11/30/2022. Using Open vSwitch with DPDK. https://github.com/openvswitch/ovs/blob/master/Documentation/howto/dpdk.rst.
[58] Richard Sanger, Brad Cowie, Matthew Luckie, and Richard Nelson. 2018. Characterising the Limits of the OpenFlow Slow-Path. In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*.
[59] Muhammad Shahbaz, Lalith Suresh, Jennifer Rexford, Nick Feamster, Ori Rottenstreich, and Mukesh Hira. 2019. Elmo: Source Routed Multicast for Public Clouds. In *ACM SIGCOMM*.
[60] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-Envisioning Remote Memory Access for Multi-Tenant Datacenters. In *ACM SIGCOMM*.
[61] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable Packet Scheduling at Line Rate. In *ACM SIGCOMM*.
[62] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. Taurus: A Data Plane Architecture for per-Packet ML. In *ASPLOS*.
[63] David E. Taylor and Jonathan S. Turner. 2007. ClassBench: A Packet Classification Benchmark. *IEEE/ACM Transations on Networking* (2007).
[64] Tigera. last accessed: 11/30/2022. Project Calico. https://www.tigera.io/project-calico/.
[65] Jean Tourrilhes, Justin Pettit, et al. last accessed: 11/30/2022. OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06). https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.
[66] William Tu, Yi-Hung Wei, Gianni Antichi, and Ben Pfaff. 2021. Revisiting the Open VSwitch Dataplane Ten Years Later. In *ACM SIGCOMM*.
[67] VMware. last accessed: 11/30/2022. VMware NSX: Network Virtualization Platform. https://www.vmware.com/products/nsx.html.
[68] VMware. last accessed: 11/30/2022. VMware's per-CPU Pricing Model. https://news.vmware.com/company/cpu-pricing-model-update-feb-2020.
[69] Open vSwitch. last accessed: 11/30/2022. Open vSwitch Manual. http://www.openvswitch.org/support/dist-docs/ovs-vswitchd.conf.db.5.pdf.
[70] Yong Wang, Boon Ang, Guolin Yang, and Wengyi Jiang. last accessed: 11/30/2022. BFD Offload in Virtual Network Interface Controller. https://patents.google.com/patent/US11196651B2/en?oq=%2316%2f661%2c879.
[71] Xilinx. last accessed: 11/30/2022. Alveo SN1000 SmartNICs. https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/sn1000-product-brief.pdf.
[72] Xilinx. last accessed: 11/30/2022. Alveo U25 SmartNIC. https://www.xilinx.com/products/boards-and-kits/alveo/u25.html.
[73] Xilinx. last accessed: 11/30/2022. OVS Offload. https://www.xilinx.com/publications/solution-briefs/partner/vvdn-ovs-solution-brief.pdf.
[74] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. 2021. ACC: Automatic ECN Tuning for High-Speed Datacenter Networks. In *ACM SIGCOMM*.

[75] Liangcheng Yu, John Sonchack, and Vincent Liu. 2020. Mantis: Reactive Programmable Switches. In *ACM SIGCOMM*.
[76] Lihua Yuan. last accessed: 11/30/2022. SONiC: Software for Open Networking in the Cloud. https://conferences.sigcomm.org/events/apnet2018/slides/lihua.pdf.