Gigaflow: Pipeline-Aware Sub-Traversal Caching for Modern SmartNICs

Annus Zulfiqar
University of Michigan
Ann Arbor, MI, USA

Ali Imran
University of Michigan
Ann Arbor, MI, USA

Venkat Kunaparaju
Purdue University
West Lafayette, IN, USA

Ben Pfaff
Feldera Inc.
Palo Alto, CA, USA

Gianni Antichi
Politecnico di Milano
Milan, Italy

Muhammad Shahbaz
University of Michigan
Ann Arbor, MI, USA

Abstract

The success of modern public/edge clouds hinges heavily on the performance of their end-host network stacks if they are to support the emerging and diverse tenants' workloads (e.g., distributed training in the cloud to fast inference at the edge). Virtual Switches (vSwitches) are vital components of this stack, providing a unified interface to enforce high-level policies on incoming packets and route them to physical interfaces, containers, or virtual machines. As performance demands escalate, there has been a shift toward offloading vSwitch processing to SmartNICs to alleviate CPU load and improve efficiency. However, existing solutions struggle to handle the growing flow rule space within the NIC, leading to high miss rates and poor scalability.

In this paper, we introduce Gigaflow, a novel caching system tailored for deployment on SmartNICs to accelerate vSwitch packet processing. Our core insight is that by harnessing the inherent pipeline-aware locality within programmable vSwitch pipelines—defining policies (e.g., L2, L3, and ACL) and their execution order (e.g., using P4 and OpenFlow)—we can create cache rules for shared segments (sub-traversals) within the pipeline, rather than caching entire flows. These shared segments can be reused across multiple flows, resulting in higher cache efficiency and greater rule-space coverage. Our evaluations show that Gigaflow achieves up to a 51% improvement in cache hit rate (average 25% improvement) over traditional caching solutions (i.e., Megaflow), while capturing up to 450× more rule space within the limited memory of today's SmartNICs—all while operating at line speed.

CCS Concepts: • **Networks** → **Packet classification**; *Programming interfaces*; **Cloud computing**; **Programmable networks**; **In-network processing**; *Data center networks*; **Network adapters**; • **Hardware** → Networking hardware.

This work is licensed under a Creative Commons Attribution 4.0 International License.

ASPLOS '25, Rotterdam, Netherlands
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1079-7/2025/03
https://doi.org/10.1145/3676641.3716000

Keywords: SmartNICs; Slowpath; Megaflow; Gigaflow; Traversal; Sub-traversal; Caching; SDN; OVS; P4; FPGA; LTM

ACM Reference Format:
Annus Zulfiqar, Ali Imran, Venkat Kunaparaju, Ben Pfaff, Gianni Antichi, and Muhammad Shahbaz. 2025. Gigaflow: Pipeline-Aware Sub-Traversal Caching for Modern SmartNICs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3676641.3716000

1 Introduction

Since the late 90s, the end-host networking stack in datacenter networks has transformed into a switching substrate built around virtual switches (vSwitches) [33, 56]. These vSwitches act as the last-hop layer in the modern distributed computing landscape—routing traffic to and from virtual machines (VMs) and containers, connecting them to the outside world [20, 33, 56, 75]. Early incarnations of these vSwitches primarily resulted in mimicking the functionalities of fixed-function hardware switches as hardcoded software switches (e.g., Linux Bridge and iptables) [21, 77]. We have come a long way since then (a) through a series of software optimizations aimed at maximizing CPU performance [56, 58, 59] to (b) leveraging hardware offloads using modern SmartNICs [25, 27, 46, 47, 83]. Nevertheless, the challenge persists: *these vSwitches struggle to scale effectively with emerging workloads and increasing link rates* [59, 90].

In software, applying the entire multi-table pipeline—comprising a sequence of transformations based on network policies (e.g., L2, L3, or ACL)—within a vSwitch for each incoming packet is prohibitively expensive. CPUs struggled with handling multiple lookups, resulting in significant performance degradation with each additional lookup [56]. Early optimizations addressed this by introducing single-lookup caches, namely Microflow and Megaflow [56]. In the Microflow cache, the first packet in a flow is processed by the multi-table pipeline to generate a single exact-match rule, which is then cached, allowing subsequent packets to bypass the full pipeline. Later, Microflow caches evolved into wildcard-based Megaflow caches, which improved aggregated vSwitch throughput by allowing a broader range of traffic patterns to be handled within the cache.

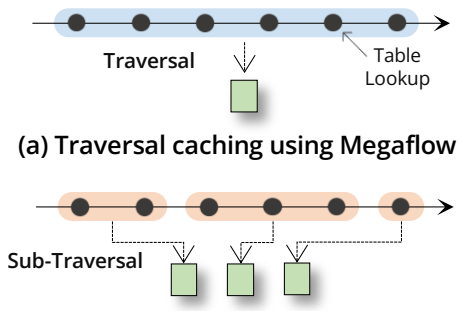


Figure 1. (a) A *traversal* is a complete sequence of table lookups through the vSwitch pipeline that generates a Megaflow rule. (b) A *sub-traversal* is a subset of these lookups within a traversal, capturing smaller, reusable segments shared across multiple flows.

More recent work has focused on enhancing lookup speeds within Megaflow caches by replacing the existing Tuple Space Search (TSS) classifier [56, 66] with compact machine learning models, such as the Range Query-Recursive Model Index (RQ-RMI) [58, 59]. Despite these improvements, CPU limitations—particularly the stagnating performance growth with the slowdown of Moore's Law [26, 42, 69, 78, 82]—continue to restrict the overall throughput of vSwitches to less than 10 Gbps per core [56, 59, 71].

There is an urgent push within the networking industry to shift from CPU-based virtual switching to SmartNICs, as evident through recent products from major players, including Amazon Nitro [1], NVIDIA Bluefield [46, 47], AMD Pensando DPU [2, 4], and Intel IPU [30]. Equipped with a hardware (HW) cache, these NICs can process and route traffic directly to and from the virtual endpoints (e.g., using SR-IOV [50]), effectively bypassing software-based processing. These NICs can reach link speeds of 400 Gbps and higher [47] when the matching rule is present in the HW cache.[1]

Yet, the main challenge with these NICs is the limited size of their HW caches, typically holding between 10–50K wildcard rules—far fewer than the software-based caches [39, 56, 74]. This limitation is attributed to the restricted power budget of SmartNICs, typically around 75 W [84], and the complexity of integrating large TCAMs on-chip [37, 45, 60, 67, 72]. As a result, despite their performance advantages, high miss rates within these caches result in significantly lower overall aggregate throughput. Thus, most implementations today use HW caches to handle only a small subset of traffic—primarily long and bursty flows—while directing the bulk of other traffic to software for processing [19, 25, 61, 74].

In this paper, we offer a fresh perspective on caching rules within SmartNICs. Traditionally, two assumptions have guided the design of caching in vSwitches: (1) multi-table lookups are prohibitively expensive, necessitating a single-lookup cache (e.g., Megaflow) [56, 59]; and (2) cache-rule generation relies solely on the locality derived from traffic patterns (i.e., temporal and spatial) [20, 56, 59, 71]. We argue that it is time to revisit these assumptions.

First, unlike CPUs, modern SmartNICs can perform multi-table lookups directly in hardware at line speeds, similar to high-performance network switches (e.g., RMT [11, 28], dRMT [16], and Trident5 [12, 13]), allowing for cross-product rule combinations that greatly expand rule-space coverage beyond physical table limits. Second, the vSwitch pipelines are programmable (e.g., using P4 [10] and OpenFlow [40]), letting operators configure not only the types of policies to apply (e.g., L2, L3, or ACL) but also their precise order. This flexibility introduces a new and powerful form of locality—*pipeline-aware locality*—that extends beyond the traditional notions of temporal and spatial locality.

In conventional caching, Microflow rules capture temporal locality by caching frequent packets from the same exact flow, while Megaflow rules use wildcards to capture spatial locality, grouping flows with overlapping headers. Pipeline-aware locality, however, leverages the structure of the vSwitch pipeline itself, enabling cache rules based on shared sequences and operations within the pipeline.

To exploit this pipeline-aware locality, we extend the concept of a *traversal* [20, 31, 33, 56, 71, 90] in vSwitches—a unique sequence of table lookups through the vSwitch pipeline that generates a Megaflow rule [31, 56, 59], reflecting both the order of table lookups and the rules matched within those tables (Figure 1a). We decompose each traversal into smaller, reusable segments, known as *sub-traversals* (Figure 1b), which represent common lookup patterns shared across flows, capturing frequently repeated sequences within the pipeline. By caching these sub-traversals and strategically installing them across SmartNIC tables, we allow multiple flows to reuse shared segments of the pipeline, creating a more efficient and scalable caching strategy.

Building on these insights—(a) line-rate multi-table lookups in SmartNICs and (b) Megaflow rules constructed from overlapping sub-traversals—we design GigaFlow, a caching system that maximizes rule-space coverage and cache hit rate while maintaining line-rate performance:

- To ensure that the sub-traversal rules in the SmartNIC tables preserve the correct sequence of the original vSwitch pipeline for a given flow, GigaFlow introduces Longest Traversal Matching (LTM) with table tags (§4.1). By selecting the longest matching sub-traversal in each table, LTM ensures the correctness of the lookup operations, with table tags maintaining the proper sequence order.
- To identify the most common sub-traversals for maximum rule-space coverage in the SmartNIC, GigaFlow leverages the disjointedness property of vSwitch pipelines (§4.2), i.e.,

[1]Note, the on-NIC ARM cores are typically less powerful and result in even poorer performance compared to the server cores [15] (see §6).

separating sub-traversals at boundaries with disjoint headers (e.g., across L2 and L3 headers). This stateless approach effectively captures pipeline-aware locality while keeping the processing cost constant, irrespective of the number of cached rules.

We implement Gigaflow as a new caching subsystem inside Open vSwitch (OVS) [56] and deploy it on P4-programmable FPGA-based SmartNICs using Xilinx's P4SDNet [6] with the OpenNIC codebase [5, 44] and Alveo U250 FPGAs [83, 86]. Our experiments use five real-world vSwitch pipelines with diverse traversal patterns (Table 1), along with Classbench [68] rulesets, and CAIDA [14] traffic traces. The results demonstrate up to 51% improvement in hit rate (25% on average), up to 90% fewer cache misses, and up to 450× greater rule-space coverage with 18% fewer cache entries, all while maintaining line-rate performance. Our Gigaflow prototype is available as open source.[2]

2 Background

Traditionally (Figure 2a), a NIC would forward all incoming packets up the host's hypervisor stack to a virtual switch (vSwitch), which applies steering policies (as flow rules) to route the packets to the destined virtual machines (VMs) or containers [20, 56]. Thus, being on the critical path demanded that the vSwitch be able to swiftly process packets with minimal overhead—as time spent processing packets here would directly impact the performance of the VMs, hosting tenant's workloads [24, 36, 43, 76, 87, 89].

2.1 Programmable vSwitch Pipelines

Early vSwitches, such as Linux Bridge [21] and iptables [77], proved inadequate in supporting advanced use cases (e.g., network virtualization [33]) crucial for modern cloud environments [56]. Hosting multiple tenants' workloads and VMs demanded sophisticated flow policies for filtering, isolating, and steering traffic across the shared infrastructure [33, 56]. For instance, data centers required a new protocol (VXLAN [79]) to partition the network into finer overlays than VLAN [80] or other traditional protocols could achieve. To express these policies, vSwitches, like Open vSwitch (OVS) [56], VMware's VDS [73], and Microsoft's Virtual Filtering Platform (VFP) [20], began exposing a programmable pipeline of match-action tables (MATs) through a standardized interface (e.g., OpenFlow [40]). Instead of relying on fixed pipelines, operators could now program these switches and specify, as a sequence of flow rules, what policies to execute and in what order.

vSwitch Caches: Microflow & Megaflow. CPUs, however, struggled to execute these programmable multi-table pipelines efficiently, leading to considerable performance degradation compared to earlier counterparts [56, 59, 90]. This led to a series of innovations in the development of fast single-lookup flow caches (such as Microflow/Megaflow in OVS [56], and Unified Flow Table (UFT) in VFP [20]), which store the results of packets processed through the multi-table pipeline (e.g., *traversal*) as flow rules. The main insight was that subsequent packets with matching header fields (exact or wildcard) could be processed directly from the flow cache, requiring only a single lookup. For example, in OVS, a hierarchy of these caches is utilized to handle incoming packets [56]. The packet is initially checked in the Microflow (or exact-match) cache, designed to capture the temporal locality of traffic (i.e., packets from a specific flow arriving frequently). Next, it is looked up in a Megaflow (or wildcard-match) cache, which leverages spatial locality (i.e., packets from flows with matching prefixes arriving closer in time). Finally, it progresses through the vSwtich multi-table pipeline, and if the instructions permit, corresponding rules are installed in the two caches. These caches function effectively when they have large sizes (i.e., to capture large flow-rule space) but quickly lose their advantage when the size is limited (see §6).

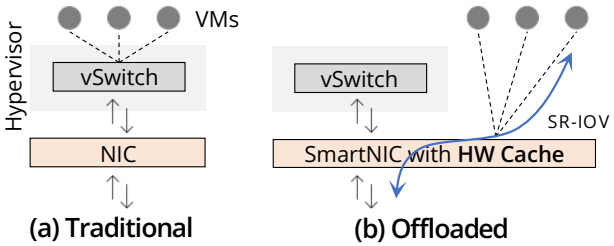


Figure 2. The modern end-host virtual switches offload traffic steering to and from VMs to SmartNICs, with hardware (HW) cache, for high-speed networking.

2.2 Accelerating vSwitches via SmartNICs

Achieving high speeds with CPUs is a tall order, especially when compounded by the need to utilize minimal CPUs, typically just one, for virtual switching in public clouds [30, 46, 47, 64, 83]. Regardless, several seminal works came forth including kernel-bypass techniques (like DPDK [18] and Netmap [70]), caching schemes (e.g., exact/wildcard matching) [56, 71], and more recent ML-based packet classifications [58, 59] to reduce the processing cycles spent by vSwitches on packet handling. Yet, despite these innovations, the deceleration of Moore's Law [42] and the intrinsic constraints of CPUs (i.e., the von Neumann bottleneck [81]), restricts the maximum achievable performance to less than 10 Gbps per CPU [56, 59, 71].

To overcome these limitations, the networking community is now turning towards SmartNICs in an effort to offload packet steering from vSwitches—taking them off the critical path [1, 2, 4, 30, 38, 41, 46, 47]. A hardware (HW) cache inside SmartNICs (Figure 2b) stores and applies steering policies to incoming packets and routes them directly to their designated endpoints (e.g., VMs and containers) using single root input/output virtualization (SR-IOV) support [50]. Unlike CPUs, these caches utilize a match-action table (MAT)

[2]Codebase: https://github.com/gigaflow-vswitch

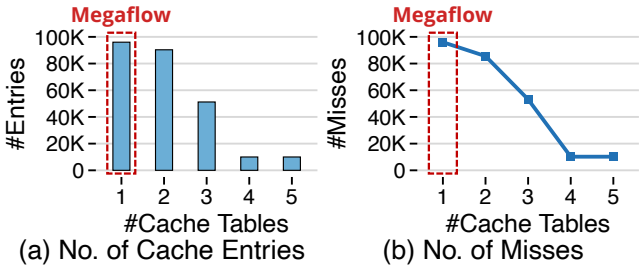


Figure 3. Increasing the cache tables (K) reduces cache entries (a) and misses (b), cutting vSwitch CPU usage by an average of 3× while covering 335× more rule space, compared to Megaflow (not shown); tested against 100,000 unique flows (details in §6).

architecture based on TCAMs [30, 47], capable of processing packets at link speeds exceeding 400 Gbps. This new configuration resembles the operation of modern switches, featuring an ASIC dedicated to line-rate packet processing, complemented by an on-switch CPU responsible for programming entries within the ASIC [30, 47, 84]. In our case, it is the vSwitch, running on the CPU (host or ARM SoC), that programs the HW cache in the SmartNIC.

The pressing need to scale end-host networking and to break past the restrictions of CPUs has, thus, resulted in the introduction of a variety of such SmartNIC products by various NIC vendors and cloud providers. These include Amazon Nitro [1], Nvidia Bluefield [46, 47], AMD Pensando DPU [2, 4], Intel IPU [30], Marvell LiquidIO [38], and Microsoft Fungible DPU [41].

However, despite their raw computing power, these SmartNICs are now constrained by the size and utilization (i.e., hit rate) of their HW caches [11, 16, 38, 39, 47, 64].

3 Motivation & Key Insights

To propel end-host networking into the era of 400 Gbps+ packet processing, it is crucial that the majority of traffic be handled by the HW caches in the SmartNICs. Doing so, however, requires a radical rethinking of how we develop and offload vSwitch caches today. Simply increasing the size of these HW caches would not suffice, as evident by most recent SmartNIC products (typically, holding 10–50K rules) [38, 39]. The limited PCIe power budget of 75 W [63, 83, 84] and the operational complexity and cost of integrating TCAMs on-chip [60], restrict the number of rule entries these HW caches can accommodate. This begs the question, *is there a way to scale the rule efficiency and performance of these SmartNICs?*

We believe there is. Until now, all our efforts have been driven by the following assumptions: (a) to improve performance, single-lookup flow caches are essential, as multi-table pipelines result in higher classification load and longer delays; and (b) to improve hit rate, the only available locality information for guiding cache-rule generation is derived exclusively from the traffic. However, two recent trends invalidate these assumptions today.

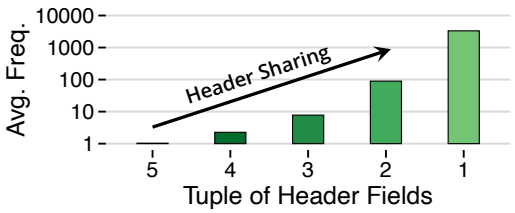


Figure 4. The average frequency of a tuple of header fields reoccurring in the Classbench ruleset (which includes 200,000 rules) increases as the number of the matching header fields decreases (5 → 1), thereby increasing the potential for header sharing.

3.1 Multi-Table Lookups & SmartNIC Offloads

Similar to data center switching ASICs (e.g., RMT [11, 28]), SmartNIC switching ASICs (e.g., on the NVIDIA Bluefield [47]) can execute multi-table lookups at line rate with sub-μs latencies [83, 84].

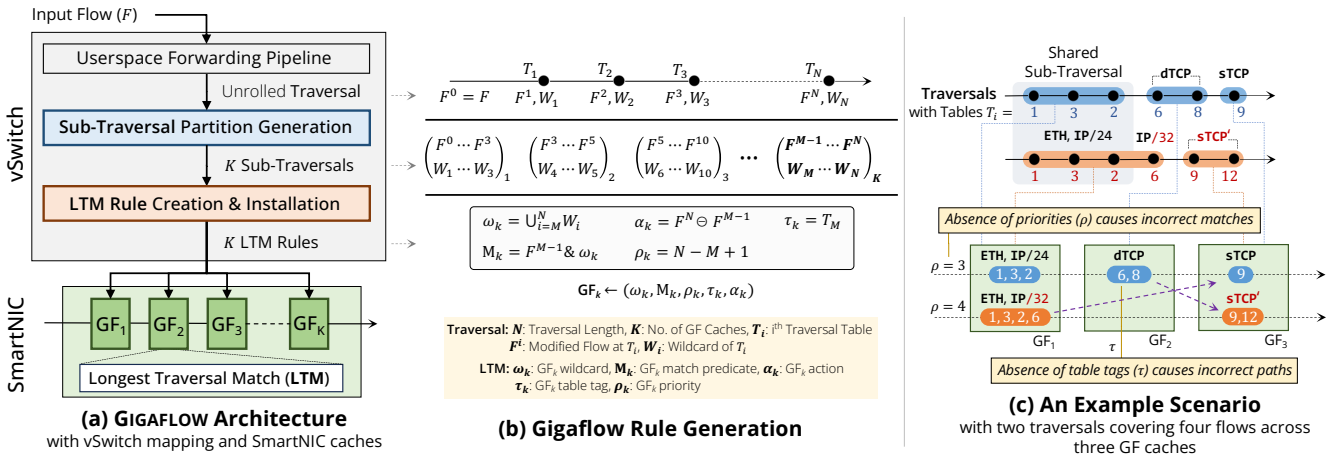
At first glance, this capability might suggest that we could simply offload the entire vSwitch pipeline onto the SmartNIC. However, several limitations make this approach infeasible: (1) the vSwitch multi-table pipelines are intricate, offering an abstraction of any-size flow tables with deep pipelines (e.g., 256 tables [56]) and complex control flow structures (e.g., loops) to support operator policies [33, 56], and (2) the low power rating and restricted chip area of SmartNICs limit how many tables they can support (typically 4–8) [30, 35, 83]. Additionally, recirculations within SmartNIC ASICs directly affect throughput and latency, resulting in sub-line-rate processing [28, 29].

***Key Insight* (I1):** Rather than directly offloading the vSwitch pipeline as-is, we can trace the control flow of a packet through the pipeline, yielding a linear sequence of lookups (i.e., a traversal). The goal is then to map this traversal (involving N tables) onto the SmartNIC pipeline with K cache tables, where K≪N.

A Megaflow cache corresponds to a mapping with K=1 (Figure 1a), which simplifies the mapping problem significantly [56]. In a Megaflow cache, each table's actions are traced and composed into a final set of actions, along with a matching wildcard, and cached as a single rule. We generalize this concept to K>1 by breaking a traversal into K sub-traversals (Figure 1b), which are then mapped across the SmartNIC's multi-table pipeline. The difficulty, however, lies in identifying optimal breakpoints for sub-traversals to maximize sharing among incoming flows. For example, using a real-world vSwitch pipeline (OLS; Table 1), Classbench ruleset [68], and CAIDA traffic traces [14], setting K=4 reduces cache misses by up to 90% (Figure 3a) and increases rule-space coverage by 335×, all while requiring only 10,000 entries (Figure 3b).

3.2 Pipeline-Aware Locality & Prog. vSwitches

The vSwitch pipelines are programmable (e.g., via P4 [10] and OpenFlow [40]); network operators can specify which



(a) Gigaflow Architecture with vSwitch mapping and SmartNIC caches

(b) Gigaflow Rule Generation

(c) An Example Scenario with two traversals covering four flows across three GF caches

Figure 5. (a, b) High-level Gigaflow architecture: the input flow (F) traverses the vSwitch userspace pipeline to produce an unrolled traversal, which is then partitioned into sub-traversals and stored as LTM rules in Gigaflow caches. (c) Example Gigaflow processing with two traversals partitioned across three caches, capturing four flows.

policies to apply and in what order—forming a sequence of table lookups through the vSwitch pipeline, with each table serving a distinct function (e.g., L2, L3, or ACL). Ideally, the most frequently and commonly used sub-sequences of these table lookups should be selected as candidate sub-traversals for installation on the SmartNIC's cache tables. However, identifying these optimal sub-traversals requires real-time awareness of traffic patterns and the cached sub-traversals; a process that becomes progressively more costly as the cache size increases.

***Key Insight* (I2):** Another approach to identifying frequently taken sub-traversals within the vSwitch pipeline is to examine the fields each table matches on. While it may seem counter-intuitive initially, breaking these sub-traversals across tables—matching on disjoint fields—could result in sub-traversals that are frequently traversed (or shared) by arriving flows.

For instance, as shown in Figure 4, a sub-traversal matching between 1 and 4 headers can be shared by about 856 flows (on average) in the Classbench [68] ruleset, which depicts a datacenter environment. However, the average sharing drops to about 1.03 when matching on the entire 5-tuple with a Megaflow cache. This indicates that breaking sub-traversals into segments of smaller, disjoint header fields would yield fewer misses and cache entries (Figure 3).

4 Design of Gigaflow

Building on these insights—caching sub-traversals from the vSwitch pipeline in the SmartNIC (**I1**) and leveraging disjointedness for pipeline-aware locality (**I2**)—we introduce Gigaflow, a novel system featuring a multi-table caching scheme within the SmartNIC, called Longest Traversal Match (LTM), §4.1. Gigaflow populates this cache by partitioning traversals into disjoint sub-traversals (with no overlapping fields), ensuring maximum rule-space coverage, §4.2.

Workflow. As illustrated in Figure 5a, Gigaflow's LTM cache in the SmartNIC consists of K feed-forward lookup tables based on the P4-programmable RMT architecture ($GF_1, GF_2, \ldots, GF_K$), which is widely supported by commercial SmartNICs today [30, 47, 64, 83, 86]. When an incoming packet misses the LTM cache, it is sent to the vSwitch, where its flow signature, F (a set of header fields), is extracted and processed through the userspace forwarding pipeline (§4.2.1). This process generates a *traversal*, which includes the accessed vSwitch pipeline tables ($T_1, T_2, \ldots, T_N$), the modified flow after each table (F^i), and a wildcard indicating the matched header bits (W_i), as shown in Figure 5b.

Next, Gigaflow divides the traversal into up to K disjoint sub-traversals, selecting a partitioning scheme that maximizes disjointness between adjacent sub-traversals (§4.2.2). These sub-traversals are converted into LTM rules, each containing a match predicate (M_k), a wildcard (ω_k), a match priority (ρ_k), a table tag (τ_k), and an action (α_k), Figure 5b. The rules are subsequently installed into the LTM cache, allowing incoming packets to leverage shared segments of the pipeline for efficient processing (§4.2.3).

For instance, Figure 5c depicts two traversals generated from different flow signatures. Both initially follow the same sequence of vSwitch tables, $T_1 = 1 \rightarrow T_2 = 3 \rightarrow T_3 = 2$, matching on common Ethernet headers (ETH) and an IP/24 prefix. They diverge at the fourth table ($T_4 = 6$), where one matches on the TCP destination port and the other on a more specific IP/32 address. By partitioning these traversals into sub-traversals, Gigaflow installs the resulting rules into the LTM cache, capturing both the original flows and potential new flows that combine segments of the original traversals (indicated by the purple paths in Figure 5c)—allowing direct handling in the SmartNIC without traversing the entire vSwitch pipeline. We will use this as a running example throughout the remainder of the paper.

4.1 GIGAFLOW SmartNIC Offload

Sub-traversal caching imposes additional challenges for ensuring accurate and consistent lookups within SmartNICs, especially in RMT-based architectures [11, 16, 28, 29]. Unlike traditional traversal caching (Figure 1a), which consolidates the entire vSwitch processing into a single, non-overlapping Megaflow rule [52, 56, 59], sub-traversal caching (Figure 1b) distributes rules across multiple cache tables. This distribution can lead to overlapping sub-traversals from different flows, resulting in multiple matches within the same table and making it harder to accurately reflect the original vSwitch traversal sequence. While Megaflow caching maintains a clear one-to-one mapping between the rule and its traversal, sub-traversal caching must handle these complexities to ensure accurate and consistent matching.

4.1.1 Longest Traversal Matching (LTM). To tackle these challenges, we introduce Longest Traversal Matching (LTM), a multi-table caching scheme for SmartNICs in GIGAFLOW. LTM performs two key tasks: (1) selecting the correct matching rule within a cache table, and (2) ensuring that the chosen sub-traversals maintain the correct sequence of the original vSwitch traversal as the packet progresses through cache tables.

In Figure 5c, a packet from the second traversal matches both rules in GF_1 since they share the same ETH headers and match both IP/24 and IP/32 prefixes. The challenge is determining which rule to select. Similarly, in GF_2, the packet could match the TCP destination port (dTCP), but this rule does not correspond to the original vSwitch traversal—thus leading to incorrect and inconsistent behavior.

LTM resolves these conflicts by selecting the rule associated with the sub-traversal that spans the most tables in the original vSwitch pipeline; hence, the name "Longest Traversal Matching." This ensures that the packet matches the most specific rule from the original traversal. In our earlier example, the packet would match the second rule (ETH and IP/32) in GF_1 as it spans more tables.

To maintain the correct traversal sequence across cache tables, LTM appends a "table ID" to the match predicate. The table ID represents the expected next table in the original vSwitch traversal. At each cache stage, the packet must match both the rule and the expected table ID, filtering out sub-traversals that are not part of the original sequence. This approach ensures consistency—causing a miss in GF_2 but correctly selecting the second rule in GF_3 for its matching TCP source port.

LTM operates similarly to Longest Prefix Matching (LPM) [32] in packet routing: while LPM selects the longest prefix for precise routing, LTM prioritizes sub-traversals that span the most tables in the vSwitch pipeline. By focusing on longer matches, LTM ensures accurate alignment with the original traversal, enabling consistent and efficient packet processing within SmartNICs.

```
table LTM_Table {
    key = {
        meta.table_tag : exact;    // Table Tag (τ)
        meta.src_port  : ternary;  // Ingress Port
        hdr.eth.smac   : ternary;  // ETH Src
        hdr.eth.dmac   : ternary;  // ETH Dst
        hdr.eth.type   : ternary;  // ETH Type
        hdr.ipv4.src   : ternary;  // IP Src
        hdr.ipv4.dst   : ternary;  // IP Dst
        hdr.ipv4.proto : ternary;  // IP Protocol
        meta.tp_src    : ternary;  // TCP Src
        meta.tp_dst    : ternary;  // TCP Dst
    }
    actions = {
        set_ethernet;              // Modify ETH Fields
        set_ip;                    // Modify IP Fields
        set_transport;             // Modify TCP Fields
        update_table_tag;          // Next Sub-Traversal
        forward;                   // Forward Pkt to Egress
        drop;                      // Drop Pkt
        NoAction;
    }
    size           = NUM_ENTRIES;
    default_action = NoAction;
}
```

Figure 6. An LTM table, as defined in P4 [10], performs an exact matches on tag τ while allowing wildcard matches on other header fields. Actions modify header fields, update the tag, and either forward or drop the packet. When a miss occurs, GIGAFLOW sends the packet to the userspace vSwitch pipeline.

4.1.2 LTM on P4-programmable RMT Architecture. GIGAFLOW utilizes the P4-based RMT architecture [11] to implement Longest Traversal Matching (LTM) in SmartNICs, using RMT's rule priority and metadata features for efficient processing. LTM assigns priorities (ρ) based on sub-traversal length, with longer sub-traversals receiving higher priority. For example, in Figure 5c, the first rule in GF_1 has a priority of $\rho = 3$, while the second has $\rho = 4$. LTM also uses P4 metadata to manage the "table ID" (i.e., tag τ), which indicates the next expected table. When a packet belonging to the second traversal exits GF_1, τ updates to 9, directing it to skip GF_2 and proceed to GF_3. This design ensures accurate traversal sequences while operating at line rate in SmartNICs.

Figure 6 illustrates an LTM table definition in P4 [10]. All LTM tables share a homogeneous structure, performing an exact match on the tag τ (8-bit metadata) and ternary matches on additional fields, such as the ingress port and all five-tuple headers. LTM rules use these wildcarded fields and match them based on the LTM priority ρ. This uniform design enables any table in the SmartNIC to support flexible sub-traversal matching without being restricted to specific vSwitch pipeline stages or networking layers, preserving the programmability of the vSwitch pipeline.

4.2 GIGAFLOW vSwitch Processing

When a packet misses the GIGAFLOW cache in the SmartNIC, it is forwarded to the vSwitch for processing. The vSwitch then executes three main steps: (1) constructing a traversal based on the vSwitch pipeline and packet's flow signature (§4.2.1), (2) partitioning the traversal into sub-traversals (§4.2.2), and (3) generating the final LTM rules for caching in the SmartNIC (§4.2.3).

4.2.1 Building vSwitch Pipeline Traversal.

To construct the traversal, the vSwitch gathers information as the packet moves through its pipeline (Figure 5b). It collects the incoming and modified outgoing flows (F^i), the sequence of table IDs (T_i) processed, and the matching wildcards (W_i).

The traversal is composed of three vectors: **T** for table IDs, **F** for modified flows after each lookup, and **W** for matched fields. Additionally, the vSwitch identifies available Gigaflow caches for mapping, represented as $\mathbf{GF} = \{GF_k \mid GF_k \text{ is not full}, k \in [1, K]\}$. Thus, the complete traversal vector is defined as $< \mathbf{T}, \mathbf{F}, \mathbf{W}, \mathbf{GF} >$.

4.2.2 Generating Sub-Traversal Partitions.

This stage takes the traversal vector $< \mathbf{T}, \mathbf{F}, \mathbf{W}, \mathbf{GF} >$ and partitions it into sub-traversals, selecting the configuration with the highest pair-wise disjointedness between adjacent sub-traversals.

> ***Disjointedness Property:*** Two sub-traversals are considered disjoint if they have no matching fields in common. For example, if one sub-traversal matches on Ethernet headers while another matches on TCP ports, they are disjoint, as in Figure 5c. When creating cache entries, it is best to place disjoint sub-traversals in separate tables and merge those that share matching fields (Figure 4). This approach maximizes the number of independent rule combinations, allowing broader coverage of flow space and improving cache efficiency, as we evaluate in §6.

To illustrate sub-traversal partition generation and selection, Figure 7 shows potential partitions for the first traversal from Figure 5c that Gigaflow explores to find the optimal partition. The disjoint field boundaries in the figure separate ETH, IP/24, and TCP source/destination ports.

A disjoint set of sub-traversals can be seen as grouping vSwitch pipeline table rules with overlapping matching fields. In the optimal partition (the last one in Figure 7), the first sub-traversal matches Ethernet/IP headers, the second targets TCP destination ports, and the third handles TCP source ports. In contrast, sub-optimal partitions cross the disjoint field boundaries (highlighted in red), combining disjoint fields within the same sub-traversal.

Algorithm. This observation leads to an effective method for identifying optimal sub-traversals: explore all possible partitions and evaluate each sub-traversal for tables with overlapping fields. If fields overlap, assign a score equal to the sub-traversal's length (i.e., the number of tables it spans); if fields are disjoint, assign a score of 0. The total score for a partition is the sum of its sub-traversal scores, with the partition achieving the highest score selected as optimal. A dynamic program can do this exploration in O(N·K) time, N being the traversal length and K being the number of Gigaflow tables.

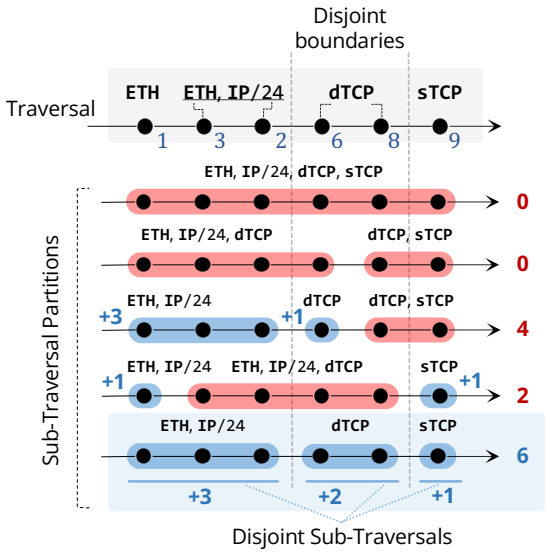


Figure 7. An example traversal showing disjoint field boundaries and various sub-traversal partitions. Partitions crossing the boundary (in red) receive a score of 0, while those within the boundary are scored by their length. The partition with the highest total score represents the most disjoint configuration.

The optimal partition has two key characteristics: (1) it maximizes disjointedness by separating distinct field sets and (2) it prioritizes longer sub-traversals, reducing the number of Gigaflow cache entries needed per traversal. In Figure 7, the optimal partition achieves a score of 6, compared to other alternatives. We further show in §6 that many real-world vSwitch pipelines (Table 1) offer similar opportunities for creating disjoint sub-traversals.

4.2.3 Creating and Installing LTM Rules.

Gigaflow's partition generator produces one partition with up to K sub-traversals, each converted into an LTM cache entry. To create Gigaflow entries, the matching wildcard (ω_k) is generated by performing a bitwise union (OR) of wildcards (W_i) from all tables in the sub-traversal. The match predicate (M_k) is constructed through a bitwise intersection (AND) of this wildcard with the initial flow at the start of the sub-traversal. The actions (α_k) are defined by calculating the commit, which represents the differences between the initial flow (F^i) and the final flow state after the sub-traversal, recorded as setfield actions in the LTM cache.

LTM sets the priority (ρ_k) of each sub-traversal rule based on its length (§4.1). The match predicate is updated to include an exact match on the table tag (τ_k), which corresponds to its starting vSwitch pipeline table ID (T_i). The actions also update the table tag to the ID of the next expected table, ensuring that packets follow the correct traversal sequence.

Managing vSwitch Pipeline Rule Dependencies. In Gigaflow, a cache hit must reflect the highest-priority traversal path in the vSwitch pipeline. To ensure this, Gigaflow

Pipeline	Description	Tables	Traversals
OFD	Openflow Data Plane Abstraction (OFDPA) [17] provides integration of HW/SW switches in CORD.	10	5
PSC	An L2L3-ACL OVS pipeline as used in Pisces [65].	7	2
OLS	OVN logical switch [55] manages virtual network topologies with logical segments using OVS.	30	23
ANT	Antrea [7–9] implements networking and security policies using OVS for a Kubernetes cluster.	22	20
OTL	Openflow Table Type Patterns (TTP) [51] for configuring L2L3-ACL policies in OVS.	8	11

Table 1. Real-world Open vSwitch (OVS) pipelines, depicting vSwitch tables and unique traversals.

combines LTM priorities (ρ) with additional field bits added to the wildcard of each cache entry, similar to Megaflow [56], preventing matches with higher-priority rules.

For instance, consider a packet destined for 192.168.21.27 with vSwitch rules prioritized by IP prefixes: (400: 192.168.14.15/32), (300: 192.168.14.0/24), (200: 192.168.0.0/16), and (100: 192.0.0.0/8). As the packet misses the first two rules but matches the third, Gigaflow adjusts the wildcard to account for the highest-priority rule that could still match, resulting in a wildcard of 255.255.240.0. The final cache entry becomes 192.168.21.27 AND 255.255.240.0 → 192.168.16.0/20. By adding these bits to the cache entry's wildcard, Gigaflow ensures that packets match only the target vSwitch rule for this flow, preventing any simultaneous matches with higher-priority rules and effectively blocking lower-priority hits.

4.3 Gigaflow Revalidation and Updates

Gigaflow handles cache revalidation through two mechanisms: (1) evicting entries when vSwitch pipeline rules are updated and (2) removing expired entries when incoming traffic no longer utilizes them.

4.3.1 Eviction Due to vSwitch Pipeline Rule Updates. When vSwitch pipeline rules change, Gigaflow revalidation identifies inconsistent entries without immediate rule pushes to the cache. Instead, it uses revalidation cycles, where Gigaflow picks the table tag (τ) of an entry, sends its parent flow to the corresponding vSwitch pipeline table, and validates it up to the length of its sub-traversal. If the actions and match predicate of the revalidated entry differ from the stored Gigaflow entry, it is marked for eviction.

4.3.2 Eviction Due to Timeouts. Gigaflow uses a *max-idle* parameter, similar to OVS [56], to remove stale entries. However, instead of evicting entire parent traversals, Gigaflow selectively evicts only stale sub-traversals. Caching sub-traversals enables faster revalidation, as sub-traversals are shorter than complete traversals, reducing revalidation overhead. This makes Gigaflow more efficient than traditional traversal caches like Megaflow in OVS (§6.3).

5 Implementation

We implement Gigaflow as a caching subsystem within the widely-used Open vSwitch (OVS) [56], adding support for both a software backend and SmartNIC offload. Using OVS commit:4f933301 [23], we modified approximately 10,000 lines of code (LoCs) across 45 files, written in C/C++. About 30% of these changes are dedicated to Gigaflow's sub-traversal partitioning and rule-creation engine. Moreover, our software backend utilizes the OVS DPDK framework (v21.11.0) [22], with an additional 10% LoCs.

For the SmartNIC offload, we used production-grade Xilinx OpenNIC [5] implementation with NetFPGA-PLUS [44] (commit:f03b61c1) with the P4-programmable FPGA-based SmartNIC—Alveo U250—as the hardware backend. The Xilinx Alveo series are FPGA-powered SmartNICs optimized for datacenter acceleration [83, 86]. We wrote 350 lines of P4 [10] code, compiling it to Verilog using the P4SDNet toolchain (v2020.1) [6] and Vivado SDK (v2020.2) [3], provided by Xilinx, to create Gigaflow's caching pipeline with four match-action tables (MATs). Each table performs ternary matches on ten standard network header fields (e.g., Ethernet, IP, TCP/UDP) and includes an exact-match field for the table tag (τ), along with priorities (ρ) and actions (e.g., to modify the tag).

Overall, our SmartNIC implementation utilizes 47% of the FPGA's lookup tables (LUTs), 33% of flip-flops (FFs), and 49% of on-chip memory (BRAM/URAM). The setup consumes 38 W on-chip power[3] and is synthesized for 100 G line-rate performance. Finally, to update SmartNIC tables at runtime, we incorporated PCIe read/write routines—generated by P4SDNet for our P4 program—into the OVS offload APIs (i.e., queue_netdev_flow_put and rte_flow) [22, 23], taking around 700 LoCs.

6 Evaluation

We evaluate Gigaflow's end-to-end cache efficiency and performance (§6.2) and conduct microbenchmarks (§6.3).

6.1 Experiment Setup

Testbed Environment. Our testbed consists of two servers connected back-to-back, serving as a device under test (DUT) and a traffic generator, respectively. These servers are equipped with a 64-core Intel Xeon Platinum 8358P CPU @ 2.60 GHz with 512 GB RAM, running Proxmox VE (v8.0.4) [57]. Each of these hosts a dual-port Intel XL710 10/40G NIC as well as Nvidia's Connectx-6 100G NIC and a Bluefield-2 DPU (with ARM SoC). The DUT server also comes equipped with a Xilinx Alveo U250 FPGA for testing our Gigaflow prototype. All these NICs and FPGAs are

[3]Reported from the Xilinx Vivado SDK (v2020.2) post-implementation thermal and power report (junction temperature set at 95 °C, typical for SmartNICs [54, 62]). The deviation from tapeout is within ±25% [85].

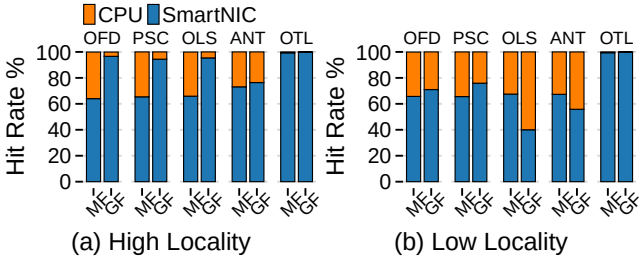


Figure 8. End-to-end cache hit rate: Gigaflow (4x8K) vs. Megaflow (32K) in high/low locality environments.

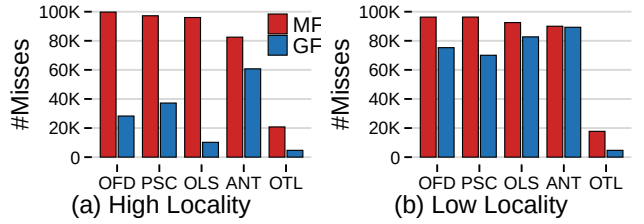


Figure 9. End-to-end cache misses: Gigaflow (4x8K) vs. Megaflow (32K) in high/low locality environments.

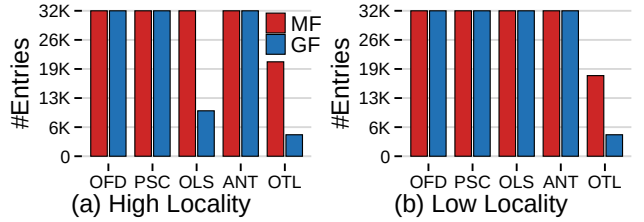


Figure 10. End-to-end cache entries: Gigaflow (4x8K) vs. Megaflow (32K) in high/low locality environments.

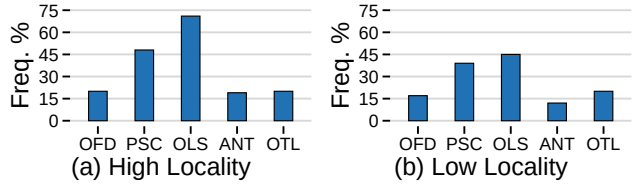


Figure 11. Frequency of sub-traversals reoccurring in Gigaflow (4x8K).

connected using a P4 Tofino-1 switch [28] to connect them depending upon the configured baseline implementations.

Baseline Configurations. Our baselines include various configurations of Open vSwitch (OVS) drawn from the standard industry deployments [48, 49, 53]: OVS/Kernel, OVS/DPDK, and OVS/Megaflow-Offload; including our Gigaflow implementation, OVS/Gigaflow-Offload. We evaluate the OVS Kernel and DPDK baselines on both the host CPU and the ARM SoC on the Nvidia DPU, while the Megaflow and Gigaflow offloads are tested using the Alveo U250 FPGA. Additionally, we examine Megaflow and Gigaflow performance using two different search algorithms: Tuple Space Search (TSS) and Neuvomatch (NM) [58, 59]. We provision a single CPU core for vSwitch software processing for all configurations, but the trend remains consistent with more CPU cores (Appendix A).

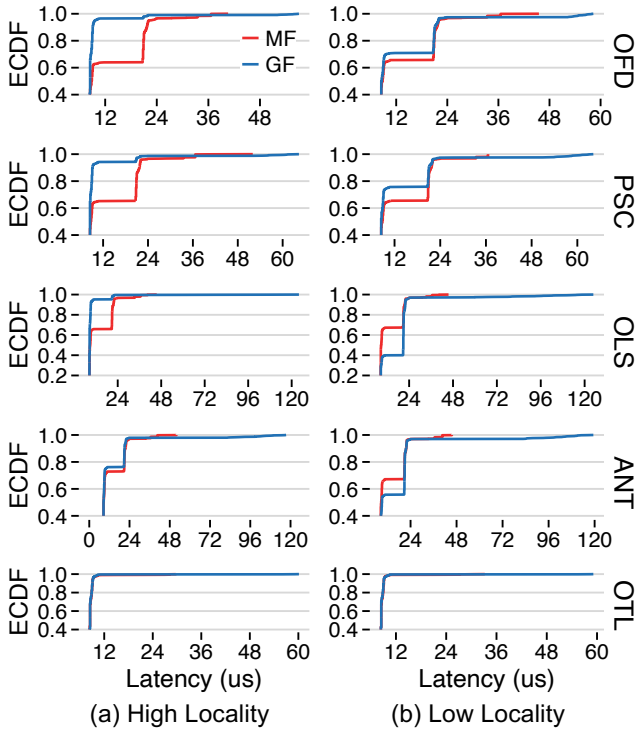


Figure 12. End-to-end latency: Gigaflow (4x8K) vs. Megaflow (32K) in high/low locality environments.

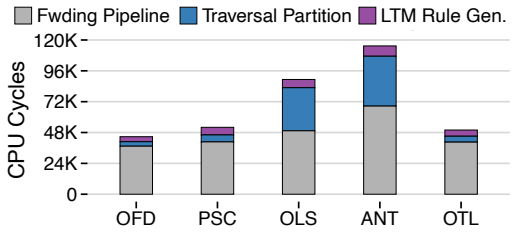


Figure 13. Average CPU cycle breakdown of vSwitch processing elements: userspace forwarding pipeline, sub-traversal partitioning, and LTM rule generation.

Real-World vSwitch Pipelines. Table 1 lists the real-world pipelines used in our evaluations, each consisting of a few to dozens of tables and varying numbers of unique traversals. Using these pipelines, we analyze Gigaflow's ability to capture pipeline-aware locality and the potential for sub-traversal sharing in real-world scenarios.

Multi-Table Rulesets and Traffic Generation. We develop a tool, Pipebench, which uses real-world pipelines (Table 1) to generate multi-table rulesets and traffic traces, featuring both high and low locality traffic (e.g., with varying access patterns to shared sub-traversals). To accomplish this, we use Classbench [68], that provides a comprehensive list of rules with fully-populated wildcard header fields for Firewall, ACL, and IPSec policies in datacenter environments. To generate the multi-table ruleset, we first randomly select a traversal that includes tables and their matching fields for a specific pipeline. Next, we randomly choose a rule from

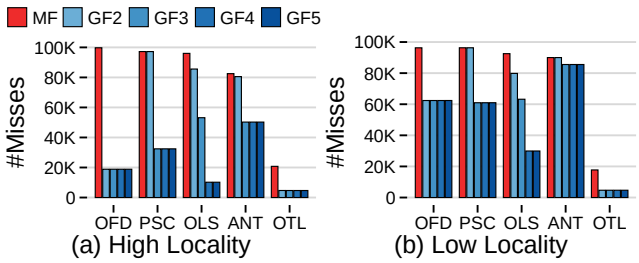


Figure 14. Cache misses with increasing number of Gigaflow tables (2–5): a 100K entry limit per table.

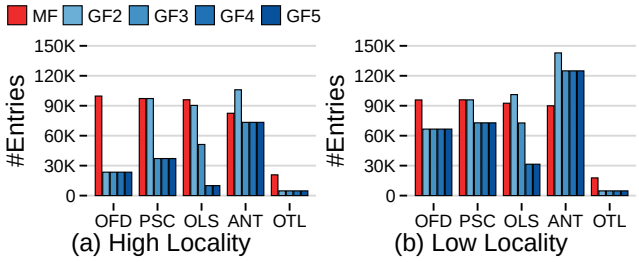


Figure 15. Cache entries with increasing number of Gigaflow tables (2–5): a 100K entry limit per table.

Classbench and map its fields to each table in the traversal. This process is repeated to generate the complete ruleset.

Similarly, we sample rules from Classbench to generate two different traffic patterns using the CAIDA [14] traffic characteristics (i.e., flow sizes and inter-packet gaps) [59]. The first pattern is created by uniformly selecting rules and modifying the CAIDA packet headers accordingly to reflect *low-locality traffic*. The second pattern selects rules based on the recurring frequency of a header tuple (Figure 4), producing *high-locality traffic* that increases the opportunity for sharing sub-traversals between flows.

6.2 End-to-End Analysis

6.2.1 Cache Behavior: Hit Rates, Misses, and Entries. In high-locality environments, Gigaflow consistently outperforms Megaflow across all five vSwitch pipelines (Table 1), achieving up to 51% higher hit rates (25% on average, Figure 8) and reducing cache misses by up to 90% (64% on average, Figure 9). These improvements stem from sub-traversal sharing, which leverages pipeline-aware locality. For instance, in the PSC pipeline, partitioning traversals by separating L2 (Ethernet) processing from ACL (IP/TCP) creates more sharing opportunities, resulting in higher hit rates. The extent of sub-traversal sharing depends on both the pipeline's design and the nature of incoming traffic (Figure 11). In low-locality traffic, with fewer sharing opportunities, hit rates decrease, and cache misses increase. However, Gigaflow still maintains performance comparable to Megaflow in pipelines, such as OFD, PSC, and OTL.

Sub-traversal sharing also affects cache utilization. In high-locality traffic, Megaflow uses 93% of available cache space, whereas Gigaflow uses only 76% on average (Figure 10). Even in low-locality environments—the worst-case scenario for sharing—Gigaflow still maintains comparable performance to Megaflow. Although sharing frequency decreases by an average of 25% (Figure 11), Gigaflow remains competitive. For OLS and ANT, the reduced sharing opportunities and higher cache usage slightly diminish Gigaflow's advantage over Megaflow in low-locality settings.

	OFD	PSC	OLS	ANT	OTL
Megaflow	32K	32K	32K	32K	32K
Gigaflow	14.7M	4.9M	10.8M	1.3M	48K

Table 2. Gigaflow (4x8K) vs. Megaflow (32K) maximum rule-space coverage with high-locality.

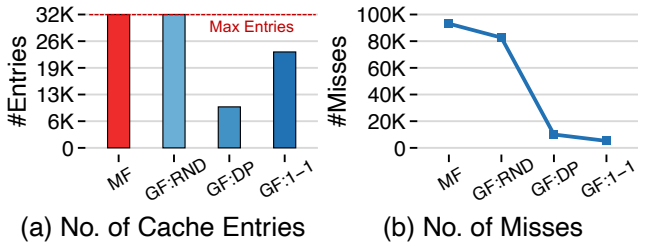


Figure 16. Comparing Gigaflow (4x8K) with partitioning schemes: Random (RND), Disjoint Paritioning (DP), and 1-1 mapping (1-1), using the OLS pipeline.

6.2.2 System Performance: Latency and CPU Usage. In high-locality environments, Gigaflow achieves significant reductions in average per-packet latency compared to Megaflow, with improvements of 29.14% for the OLS pipeline, 31% for OFD, and 27% for PSC (Figure 12). Although both Gigaflow and Megaflow offloads, using our FPGA-based SmartNIC, have the same hardware cache hit latency of about 9 μs, Gigaflow's higher cache hit rate minimizes the need to traverse the vSwitch multi-table pipeline, resulting in lower overall latency. In low-locality scenarios, where cache hits are less frequent, the latency improvements are more modest—6% for PSC, 1.67% for OFD, and 0.02% for OTL. Larger pipelines, like OLS and ANT, see higher latencies due to increased cache misses and the overhead associated with vSwitch processing, including pipeline lookups, sub-traversal partitioning, and LTM rule generation.

The CPU usage breakdown for vSwitch processing (Figure 13) reveals that larger pipelines, such as OLS and ANT, experience additional overhead when using Gigaflow; the sub-traversal partitioning and LTM rule generation add 80% and 68% more processing time on top of the userspace forwarding pipeline, which Megaflow does not incur. In contrast, smaller pipelines like PSC, OTL, and OFD exhibit lower overheads of 28%, 23%, and 20%, respectively. Despite the added complexity, the substantial reduction in cache misses helps Gigaflow offset these overheads, particularly in high-locality settings (Figure 12).

6.3 Microbenchmarks

6.3.1 Gigaflow's performance improves with the addition of more SmartNIC tables, leading to fewer cache

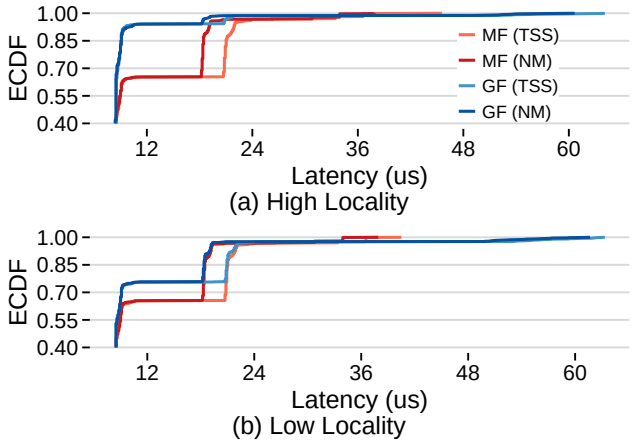


Figure 17. Comparison of Megaflow and Gigaflow with two search algorithms: original Tuple Space Search (TSS) [56, 66] and Nuevomatch (NM) [58, 59].

misses and reduced cache entries. As shown in Figures 14 and 15, increasing the number of SmartNIC tables from 1 (Megaflow) to 5, each containing 100K entries, enhances Gigaflow's performance in both high- and low-locality environments. Different pipelines benefit from varying numbers of tables—OFD fully utilizes its disjointedness with just two tables, while PSC sees gains up to three tables. Larger pipelines, like OLS, continue to benefit up to four tables, where additional partitioning opportunities arise. However, adding more tables also increases vSwitch processing overhead, as the number of potential sub-traversal partitions to explore grows. Still, the overhead remains within 200 μs even for the larger pipeline, like OLS and ANT.

6.3.2 Gigaflow achieves orders of magnitude more rule space coverage than Megaflow with the same number of cache entries. Using 4x8K tables, Gigaflow captures up to two orders of magnitude more rule space across all pipelines compared to Megaflow with 32K entries (same total cache size), Table 2. This is enabled by Gigaflow's shared sub-traversal caching, which allows cross-product rule combinations across multiple tables, vastly increasing coverage. For example, pipelines like OFD, OLS, and PSC see 459×, 337×, and 156× greater rule space coverage, respectively. Even pipelines with less partitioning potential, such as ANT and OTL, show improvements of 40× and 1.5×.

6.3.3 Gigaflow's disjoint partitioning (DP) achieves performance close to an ideal 1-1 mapping approach, but with 2.8× fewer cache entries. As shown in Figure 16, using the OLS pipeline with 100K unique flows and Gigaflow (4x8K), a random partitioning approach (RND) reduces cache misses by 11% compared to Megaflow, while consuming the entire cache. In contrast, disjoint partitioning (DP) reduces cache misses by 89% while using just 31% of the cache entries. We also compare against an ideal approach (1-1 mapping), where we assume each table in the traversal has a corresponding table in the SmartNIC. With this 1-1 mapping, cache misses are reduced by 94%, slightly better than disjoint partitioning, but consumes 2.8× more entries.

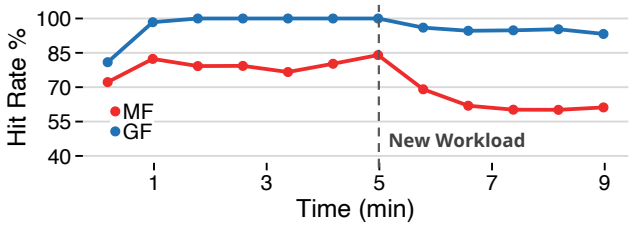


Figure 18. End-to-end cache hit rate with dynamic workloads: Gigaflow (4x8K) vs. Megaflow (32K) operating in a high-locality environment; a new workload with 50K unique flows is introduced at time 5 min.

6.3.4 Gigaflow outperforms software cache optimizations through its extensive rule-space coverage in the SmartNIC. We evaluate Gigaflow (4x8K) against Megaflow (32K) using two software cache search algorithms—Tuple Space Search (TSS) [56, 66] and Nuevomatch (NM) [58, 59]—on the PSC pipeline with a 100K unique flow traffic trace. As shown in Figure 17, in high-locality settings, Megaflow with NM reduces average latency from 13.4 μs to 12.5 μs (a 6.8% improvement over TSS). In contrast, Gigaflow with TSS achieves 9.8 μs latency—21.6% faster than Megaflow with NM. Adding NM to Gigaflow yields a slightly further reduction to 9.65 μs.

6.3.5 Gigaflow maintains a high hit rate with dynamically arriving workloads. We evaluate Gigaflow (4x8K) against Megaflow (32K) on the PSC pipeline with dynamically arriving workloads; the userspace vSwitch rules and the incoming traffic patterns (e.g., arrival times) remain the same, as new traffic flows arrive. We test using 100K unique flows divided evenly across two workloads (50K each) in a high-locality environment. At time 5 min (Figure 18), we introduce the second workload, which causes Megaflow's hit rate to drop drastically from 84% to 61.18%, while Gigaflow sustains its performance (at 93.26%) due to significantly higher rule space coverage (Table 2). In a low-locality scenario, both caches perform similarly.

6.3.6 Gigaflow achieves the lowest latency and 2× faster cache revalidation compared to OVS baselines. Gigaflow offers the lowest cache hit latency among all OVS configurations, with both OVS/Gigaflow-Offload and OVS/Megaflow-Offload achieving 8.62 ± 0.4 μs. In contrast, OVS/DPDK on a host CPU has a higher latency of 12.61 ± 1.1 μs, while the Bluefield-DPU's ARM cores perform even slower at 51.26 ± 9.7 μs. OVS/Kernel on both the host and Bluefield-DPU show much higher latencies, at 671.48 ± 13.4 μs and 3,606.37 ± 237.1 μs, respectively. Additionally, Gigaflow revalidates caches twice as fast as Megaflow. For instance, revalidating the Megaflow cache (32K entries) with the OLS pipeline takes 527 ms, while Gigaflow (4x8K) completes the task in 272 ms, about 2× faster.

7 Limitations & Future Work

Traffic-Profile-Guided Optimizations. In low-locality environments, Gigaflow may underperform compared to Megaflow since it depends solely on vSwitch pipelines to identify sub-traversal sharing opportunities. To address this, Gigaflow can use profile-guided optimization by periodically sampling packets from the SmartNIC to assess potential sub-traversal sharing in the traffic. If sharing opportunities are limited, Gigaflow could switch to using Megaflow entries in the cache, maintaining baseline performance.

Alternative Methods for Sub-Traversal Partitioning. Gigaflow employs a disjoint partitioning (DP) algorithm to divide traversals based on field-level separation. While our results show that DP incurs up to 2× the processing cost over traditional approaches, this overhead depends on both the number of Gigaflow tables and the traversal length. With approaches like Nuevomatch [58, 59] showing benefits of machine learning in network processing, we could incorporate it in finding more efficient Gigaflow mappings (e.g., by optimizing traversal partitioning based on traffic patterns).

8 Related Work

Software Flow Caching. Tuple Space Search (TSS) [66] is widely used for implementing lookups in vSwitches for both OpenFlow tables and caches [56], operating with a cost of O(M), where M represents the number of unique masks. Nuevomatch [58, 59] introduced RQ-RMI models [34], which use small perceptron trees to predict the lookup index with bounded error, lowering the lookup cost to O(1), but without affecting the cache miss volume. Instead, Gigaflow focuses on optimizing the cache miss rate.

Systems for Cache Management. OVS [56] prevents rule overlaps by adding extra bits to the Megaflow wildcard, ensuring unique match predicates for each traversal, allowing a single Megaflow cache entry per traversal. PipeCache [88] employs multi-staged hardware TCAMs, using simplified OpenFlow models to offload popular rules via a 1-to-1 mapping. Elixir [74] dynamically adjusts caching, adapting to predicted flow burstiness by keeping elephant flows in hardware and mice flows in software. In contrast, Gigaflow optimizes disjoint sub-traversal caching on SmartNICs, enhancing cache efficiency while sustaining line rate.

9 Conclusion

We presented Gigaflow, a multi-table sub-traversal cache architecture that leverages pipeline-aware locality to efficiently capture significantly larger rule spaces on SmartNICs. By leveraging disjointedness and optimizing cache storage, Gigaflow improves cache hit rate by up to 51% (average 25%) and reduces CPU-bound cache misses by up to 90% (average 64%) compared to Megaflow cache. Gigaflow delivers these improvements without compromising the line rate. We believe Gigaflow paves the way for more effective SmartNIC deployments, offering a scalable approach to manage and accelerate network processing in emerging cloud and AI data centers.

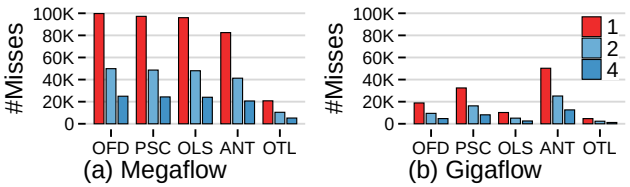


Figure 19. As the number of CPU cores for vSwitch processing increases, Gigaflow achieves performance gains similar to Megaflow.

Acknowledgments

We are grateful to our shepherd, Andrew Putnam, along with Cheng-Chun William Tu, Ori Rottenstreich, and the anonymous reviewers for their valuable feedback in strengthening this paper. We also thank Athina Terzoglou, Alon Rashelbach, and Mihai Budiu for their insightful discussions. This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; by NSF awards CAREER-2338034 and CNS-2211381; and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"). Support also came in part from VMware by Broadcom.

A Impact of CPU Core Scaling on Gigaflow Performance

As data centers become more conservative in CPU allocation for infrastructure tasks like vSwitch processing—often restricting it to a single core—we explore how Gigaflow performs with multiple CPU cores. OVS allows per-CPU distribution of cache misses from the SmartNIC using RSS, balancing misses across cores. Figure 19 shows that, like Megaflow, Gigaflow reduces cache misses per core proportionally with increased cores. However, Gigaflow achieves this with a lower total CPU load across different pipelines.

B Artifact Appendix

B.1 Abstract

The artifact includes the source code of the Gigaflow cache integrated into Open vSwitch. It is publicly available on figshare at https://doi.org/10.6084/m9.figshare.28319711, or on GitHub at https://github.com/gigaflow-vswitch.

B.2 Artifact Repositories

- OVS with Gigaflow Cache (gvs)
- Traffic Generator (tgen)
- vSwitch Pipelines (slowpath-pipelines), on figshare.

B.3 Dependencies

- **Software**: Autoconf, Automake, libtool, GNU Make, Clang (v3.4 or later), Python (v3.7 or later)

- **Hardware**: Two linux-based servers each equipped with a 16-core x86_64 CPU, 16 GB RAM, and a DPDK-supported SmartNIC.

References

[1] Amazon. AWS Nitro System. https://aws.amazon.com/ec2/nitro/, last accessed: 02/05/2025.

[2] AMD. AMD PENSANDO DSC3-400 DISTRIBUTED SERVICES CARD. https://www.amd.com/content/dam/amd/en/documents/pensando-technical-docs/product-briefs/pensando-dsc3-product-brief.pdf, last accessed: 02/05/2025.

[3] AMD. AMD Vivado Design Suite. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html, last accessed: 02/05/2025.

[4] AMD. Pensando. https://www.amd.com/en/accelerators/pensando, last accessed: 02/05/2025.

[5] AMD Xilinx. AMD OpenNIC Project. https://github.com/Xilinx/open-nic, last accessed: 02/05/2025.

[6] AMD Xilinx. Vitis Networking P4. https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html, last accessed: 02/05/2025.

[7] Antrea. Antrea: Enhance pod networking and enforce network policies for Kubernetes clusters. https://antrea.io/, last accessed: 02/05/2025.

[8] Antrea. Antrea OVS Pipeline. https://antrea.io/docs/main/docs/design/ovs-pipeline/, last accessed: 02/05/2025.

[9] Antrea-IO. Antrea OVS Pipeline. https://github.com/antrea-io/antrea/blob/main/docs/design/ovs-pipeline.md, last accessed: 02/05/2025.

[10] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming Protocol-Independent Packet Processors. In *ACM SIGCOMM CCR* (2014).

[11] Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *ACM SIGCOMM* (2013).

[12] BROADCOM. Trident 5 / BCM78800 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78800, last accessed: 02/05/2025.

[13] BROADCOM. Trident4 / BCM56880 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series, last accessed: 02/05/2025.

[14] CAIDA. The CAIDA UCSD Anonymized Internet Traces. https://www.caida.org/catalog/datasets/passive_dataset/, last accessed: 02/05/2025.

[15] Chen, X., Zhang, J., Fu, T., Shen, Y., Ma, S., Qian, K., Zhu, L., Shi, C., Liu, M., and Wang, Z. Demystifying Datapath Accelerator Enhanced Off-path SmartNIC. *arXiv preprint arXiv:2402.03041* (2024).

[16] Chole, S., Fingerhut, A., Ma, S., Sivaraman, A., Vargaftik, S., Berger, A., Mendelson, G., Alizadeh, M., Chuang, S.-T., Keslassy, I., Orda, A., and Edsall, T. dRMT: Disaggregated Programmable Switching. In *ACM SIGCOMM* (2017).

[17] Cord OF-DPA. OpenSwitch OF-DPA User Guide. https://netbergtw.com/wp-content/uploads/Files/OPS_of_dpa.pdf, last accessed: 02/05/2025.

[18] DPDK. DPDK. https://www.dpdk.org/, last accessed: 02/05/2025.

[19] Durner, R., and Kellerer, W. Network Function Offloading Through Classification of Elephant Flows. *IEEE Transactions on Network and Service Management* (2020).

[20] Firestone, D. VFP: A Virtual Switch Platform for Host SDN in the Public Cloud. In *USENIX NSDI* (2017).

[21] Foundation, T. L. Linux Bridge. https://wiki.linuxfoundation.org/networking/bridge, last accessed: 02/05/2025.

[22] Github. DPDK. https://github.com/DPDK/dpdk, last accessed: 02/05/2025.

[23] Github. Open vSwitch. https://github.com/openvswitch/ovs, last accessed: 02/05/2025.

[24] Gobieski, G., Lucia, B., and Beckmann, N. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *ACM ASPLOS* (2019).

[25] Gupta, M. 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, last accessed: 02/05/2025.

[26] Hennessy, John L. and Patterson, David A. *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. 2017.

[27] Intel. 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, last accessed: 02/05/2025.

[28] Intel. 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, last accessed: 02/05/2025.

[29] Intel. 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, last accessed: 02/05/2025.

[30] Intel. Intel Infrastructure Processing Units (IPUs) and SmartNICs. https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html, last accessed: 03/12/2024.

[31] Jackson, E. J., Walls, M., Panda, A., Pettit, J., Pfaff, B., Rajahalme, J., Koponen, T., and Shenker, S. Softflow: a middlebox architecture for open vSwitch. In *USENIX ATC* (2016).

[32] Juniper Networks. Longest Prefix Matching in Networking Chips. https://community.juniper.net/blogs/sharada-yeluri/2023/01/02/longest-prefix-matching-in-networking-chips, last accessed: 02/05/2025.

[33] Koponen, T., Amidon, K., Balland, P., Casado, M., Chanda, A., Fulton, B., Ganichev, I., Gross, J., Ingram, P., Jackson, E., Lambeth, A., Lenglet, R., Li, S.-H., Padmanabhan, A., Pettit, J., Pfaff, B., Ramanathan, R., Shenker, S., Shieh, A., Stribling, J., Thakkar, P., Wendlandt, D., Yip, A., and Zhang, R. Network Virtualization in Multi-tenant Datacenters. In *USENIX NSDI* (2014).

[34] Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The Case for Learned Index Structures. In *ACM SIGMOD* (2018).

[35] Kummrow, P. 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, last accessed: 02/05/2025.

[36] Lao, C., Le, Y., Mahajan, K., Chen, Y., Wu, W., Akella, A., and Swift, M. ATP: In-network Aggregation for Multi-tenant Learning. In *USENIX NSDI* (2021).

[37] Lin, M., Luo, J., and Ma, Y. A Low-Power Monolithically Stacked 3D-TCAM. In *2008 IEEE International Symposium on Circuits and Systems (ISCAS)* (2008).

[38] Marvell. Data Processing Units (DPU). https://www.marvell.com/products/data-processing-units.html, last accessed: 02/05/2025.

[39] Marvell. Marvell LiquidIO III. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf, last accessed: 02/05/2025.

[40] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: Enabling Innovation in Campus Networks. In *ACM SIGCOMM CCR* (2008).

[41] Microsoft. Microsoft announces acquisition of Fungible to accelerate datacenter innovation. https://blogs.microsoft.com/blog/2023/01/09/microsoft-announces-acquisition-of-fungible-to-accelerate-

datacenter-innovation/, last accessed: 02/05/2025.
[42] MIT CSAIL Alliances. The Death of Moore's Law: What it means and what might fill the gap going forward. https://cap.csail.mit.edu/death-moores-law-what-it-means-and-what-might-fill-gap-going-forward, last accessed: 02/05/2025.
[43] Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. PipeDream: generalized pipeline parallelism for DNN training. In *ACM SOSP* (2019).
[44] NetFPGA. NetFPGA-PLUS. https://github.com/NetFPGA/NetFPGA-PLUS, last accessed: 02/05/2025.
[45] NSC by orhanergun.net. Exploring TCAM Memory. https://netsec-cloud.com/tcam-memory, last accessed: 02/05/2025.
[46] Nvidia. ConnectX-6 Dx 100G/200G Ethernet NIC. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/networking-overal-dp, last accessed: 02/05/2025.
[47] Nvidia. NVIDIA BLUEFIELD DATA PROCESSING UNITS. https://www.nvidia.com/en-us/networking/products/data-processing-unit/, last accessed: 02/05/2025.
[48] Nvidia. Nvidia Bluefield DPU BSP v3.8.5. https://docs.nvidia.com/networking/display/bluefielddpuosv385/virtual+switch+on+bluefield+dpu, last accessed: 02/05/2025.
[49] Nvidia. OVS-DOCA Hardware Offloads. https://docs.nvidia.com/doca/sdk/ovs-doca+hardware+offloads/index.html, last accessed: 02/05/2025.
[50] Nvidia. Single Root IO Virtualization - SR-IOV. https://docs.nvidia.com/networking/display/mlnxofedv461000/single+root+io+virtualization+(sr-iov), last accessed: 02/05/2025.
[51] ONF. OpenFlow Table Type Patterns. https://opennetworking.org/wp-content/uploads/2013/04/OpenFlow Table Type Patterns, last accessed: 02/05/2025.
[52] Open-vSwitch. ovs/ofproto/ofproto-dpif-upcall.c. https://github.com/openvswitch/ovs/blob/master/ofproto/ofproto-dpif-upcall.c, last accessed: 02/05/2025.
[53] Openstack. Open vSwitch with DPDK Datapath. https://docs.openstack.org/neutron/latest/admin/config-ovs-dpdk.html, last accessed: 02/05/2025.
[54] Oracle. Datacenter and Server Thermal Trends and Challenges. https://www.meptec.org/Resources/6%20-%20Oracle.pdf, last accessed: 02/07/2025.
[55] OVN. Open Virtual Network. https://www.ovn.org/en/, last accessed: 02/05/2025.
[56] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. The Design and Implementation of Open vSwitch. In *USENIX NSDI* (2015).
[57] Proxmox. Proxmox. https://www.proxmox.com/en/, last accessed: 02/05/2025.
[58] Rashelbach, A., Rottenstreich, O., and Silberstein, M. A Computational Approach to Packet Classification. In *ACM SIGCOMM* (2020).
[59] Rashelbach, A., Rottenstreich, O., and Silberstein, M. Scaling Open vSwitch with a Computational Cache. In *USENIX NSDI* (2022).
[60] Rasmussen, A., Kragelund, A., Berger, M., Wessing, H., and Ruepp, S. TCAM-based high speed Longest prefix matching with fast incremental table updates. In *IEEE International Conference on High Performance Switching and Routing (HPSR)* (2013).
[61] Sarrar, N., Uhlig, S., Feldmann, A., Sherwood, R., and Huang, X. Leveraging Zipf's law for traffic offloading. *ACM SIGCOMM CCR* (2012).
[62] Scott Schweitzer, CISSP. Power, Heat, Space, and the Move to Double-Wide SmartNICs. https://www.linkedin.com/pulse/power-heat-space-move-double-wide-smartnics-scott-schweitzer-cissp/, last accessed: 02/07/2025.
[63] Serve The Home. 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/, last accessed: 02/05/2025.
[64] Serve The Home. Pensando Distributed Services Architecture SmartNIC. https://www.servethehome.com/pensando-distributed-services-architecture-smartnic/, last accessed: 02/05/2025.
[65] Shahbaz, M., Choi, S., Pfaff, B., Kim, C., Feamster, N., McKeown, N., and Rexford, J. PISCES: A Programmable, Protocol-Independent Software Switch. In *ACM SIGCOMM* (2016).
[66] Srinivasan, V., Suri, S., and Varghese, G. Packet Classification Using Tuple Space Search. *SIGCOMM CCR* (1999).
[67] Synopsys. An Introduction to TCAMs. https://www.synopsys.com/designware-ip/technical-bulletin/introduction-to-tcam.html, last accessed: 02/05/2025.
[68] Taylor, D. E., and Turner, J. S. ClassBench: A Packet Classification Benchmark. *IEEE/ACM Transations on Networking* (2007).
[69] The Economist. Jensen Huang says Moore's law is dead. Not quite yet. https://www.economist.com/science-and-technology/2023/12/13/jensen-huang-says-moores-law-is-dead-not-quite-yet, last accessed: 02/05/2025.
[70] The Netmap Project. netmap - the fast packet I/O framework. http://info.iet.unipi.it/~luigi/netmap/, last accessed: 02/05/2025.
[71] Tu, W., Wei, Y.-H., Antichi, G., and Pfaff, B. Revisiting the Open VSwitch Dataplane Ten Years Later. In *ACM SIGCOMM* (2021).
[72] Ullah, Z., Jaiswal, M. K., Cheung, R. C., and So, H. K. UE-TCAM: An Ultra Efficient SRAM-Based TCAM. In *TENCON 2015-2015 IEEE Region 10 Conference* (2015).
[73] VMWare. vSphere Distributed Switch. https://www.vmware.com/products/vsphere/distributed-switch.html, last accessed: 02/05/2025.
[74] Wang, Y., Li, D., Lu, Y., Wu, J., Shao, H., and Wang, Y. Elixir: A High-performance and Low-cost Approach to Managing Hardware/Software Hybrid Flow Tables Considering Flow Burstiness. In *USENIX NSDI* (2022).
[75] Wei, C., Li, X., Yang, Y., Jiang, X., Xu, T., Yang, B., Wu, T., Xu, C., Lv, Y., Gao, H., Zhang, Z., Chen, Z., Wang, Z., Zhang, Z., Zhu, S., and Chen, W. Achelous: Enabling Programmability, Elasticity, and Reliability in Hyperscale Cloud Networks. In *ACM SIGCOMM* (2023).
[76] Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. TernGrad: ternary gradients to reduce communication in distributed deep learning. In *NeurIPS* (2017).
[77] Wikipedia. iptables. https://en.wikipedia.org/wiki/Iptables, last accessed: 02/05/2025.
[78] Wikipedia. Moore's law. https://en.wikipedia.org/wiki/Moores_law, last accessed: 02/05/2025.
[79] Wikipedia. Virtual Extensible LAN. https://en.wikipedia.org/wiki/Virtual_Extensible_LAN, last accessed: 02/05/2025.
[80] Wikipedia. VLAN. https://en.wikipedia.org/wiki/VLAN, last accessed: 02/05/2025.
[81] Wikipedia. von Neumann architecture. https://en.wikipedia.org/wiki/Von_Neumann_architecture, last accessed: 02/05/2025.
[82] Wired. Is Moore's Law Really Dead? https://www.wired.com/story/moores-law-really-dead/, last accessed: 02/05/2025.
[83] Xilinx. Alveo SN1000 SmartNICs. https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/sn1000-product-brief.pdf, last accessed: 02/05/2025.
[84] Xilinx. Alveo U25 SmartNIC. https://www.xilinx.com/publications/product-briefs/alveo-u25-product-brief.pdf, last accessed: 02/05/2025.
[85] Xilinx. Vivado Design Suite User Guide - Power Analysis and Optimization. https://www.xilinx.com/support/documents/sw_manuals/xilinx2021_2/ug907-vivado-power-analysis-optimization.pdf, last accessed: 03/02/2025.
[86] Xilinx, A. Alveo U250 Data Center Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u250.html, last accessed: 02/05/2025.

[87] Xu, X., Ding, Y., Hu, S. X., Niemier, M., Cong, J., Hu, Y., and Shi, Y. Scaling for edge inference of deep neural networks. In *Nature Electronics* (2018).

[88] Yang, J., Li, T., Yan, J., Li, J., Li, C., and Wang, B. Pipecache: High hit rate rule-caching scheme based on multi-stage cache tables. *Electronics* (2020).

[89] Zinkevich, M., Weimer, M., Li, L., and Smola, A. Parallelized Stochastic Gradient Descent. In *NeurIPS* (2010).

[90] Zulfiqar, A., Pfaff, B., Tu, W., Antichi, G., and Shahbaz, M. The Slow Path Needs an Accelerator Too! In *ACM SIGCOMM CCR* (2023).