XFir: Accelerating New-Flow Setup on Host Servers of a Large Cloud Network

Shihan Lin[‡*], Shunqiao Jiang[†*], Liang Wang[†], Jian Wang[†], Chao Pei[†], Jian Zhao[†], Wenjun Wu[†], Kai Ren[†], Lijun Zhuang[¶], Qingmin Liu[¶], Heng Yu[♠], Sirui Li[†], Yibo Huang[‡], Yifei Zhu[♦], Yunming Xiao[♥], Ang Chen[‡], Linghe Kong[♦], Congcong Miao[♣]

[†]Tencent [‡]University of Michigan [¶]JaguarMicro, Shenzhen [♠]Tsinghua University [♦]Shanghai Jiao Tong University
[♥]The Chinese University of Hong Kong, Shenzhen [♣]National University of Singapore

Abstract

In today's cloud networks, host servers widely deploy Data Processing Units (DPUs) as network accelerators under the "Sep-Path" paradigm. However, as server capabilities scale with increasing CPU cores and network bandwidth, the software slow path (executed on a DPU's CPU) has become a critical bottleneck for workloads with high new-flow rates. Meanwhile, new-flow setup logic on host servers must continuously evolve to meet diverse and changing customer demands, making flexibility a key requirement alongside performance. To address this gap, we present XFir, the first hardware-accelerated new-flow setup system for cloud host servers that delivers high CPS throughput while preserving sufficient flexibility. XFir leverages a next-generation DPU equipped with a Cloud Network co-Processor (CNP) to execute the host server's new-flow setup logic. XFir redesigns the host-server flow-setup datapath and table layout, optimizes LPM lookups, and introduces CPU–CNP collaboration mechanisms to further improve performance and reliability. Our evaluation shows that XFir achieves over 776K new-flow CPS on a single host server with 11.7μs slow-path latency. Compared to prior work (Fornax), XFir achieves 4.8× CPS and reduces latency by 69.2%. Moreover, XFir is cost-effective to deploy, requiring only a single DPU per host. Overall, XFir improves new-flow throughput while maintaining development flexibility at low financial cost.

CCS Concepts

• **Networks** → **Data center networks**; **Cloud computing**; • **Hardware** → **Networking hardware**.

Keywords

DPU, CPS, Slow path, Hardware acceleration, Connection setup

ACM Reference Format:

Shihan Lin, Shunqiao Jiang, Liang Wang, Jian Wang, Chao Pei, Jian Zhao, Wenjun Wu, Kai Ren, Lijun Zhuang, Qingmin Liu, Heng Yu, Sirui Li, Yibo Huang, Yifei Zhu, Yunming Xiao, Ang Chen, Linghe Kong, Congcong Miao. 2026. XFir: Accelerating New-Flow Setup on Host Servers of a Large Cloud Network. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3789240.3829131

*Both authors contributed equally to this work.

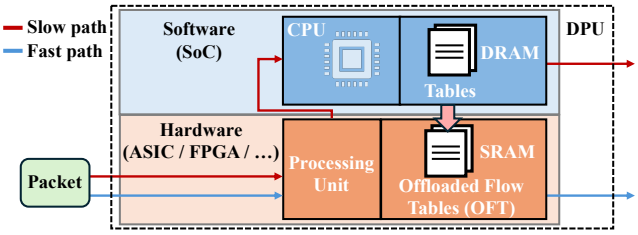


Figure 1: Sep-Path architecture of a DPU

1 Introduction

Modern cloud servers consolidate hundreds of CPU cores and hundreds of Gbps of network bandwidth to host large numbers of virtual machines (VMs). This significantly improves efficiency but severely stresses the host's packet-processing pipeline. To sustain line-rate performance, hardware accelerators (*e.g.*, DPUs [41], FPGAs [2], programmable ASICs [9]) are increasingly adopted for offloading network processing.

Meanwhile, cloud services evolve rapidly, requiring frequent updates to network logic that rigid hardware alone cannot support. To balance throughput and agility, cloud systems commonly adopt a "*Sep-Path*" architecture that splits packet processing into a programmable slow path and a high-performance fast path: the slow path runs in software (*e.g.*, a DPU's CPU or a server's CPU) to enable flexible updates but cannot sustain high traffic volumes, while the fast path executes in hardware accelerators to deliver line-rate processing with limited programmability [68, 69].

Figure 1 shows a typical Sep-Path design for a DPU on a host server [68]. The slow path handles ***new-flow setup*** [1], which we define as performing the required network functions via a sequence of table lookups, such as routing, Access Control Lists (ACLs), and Security Groups (SGs), upon receiving or sending a new flow's initial packets. The results are assembled into an entry and inserted into the fast path's offloaded flow table (OFT). Subsequent packets in the same flow are then processed entirely in the fast path at high speed, without further software involvement. Meanwhile, developers can reprogram the slow-path logic to update the table-lookup workflow. This collaboration of two paths accelerates steady-state packet processing and preserves flexibility to evolve functionalities.

However, the Sep-Path design implicitly caps the speed of new-flow setup to software capabilities. As the number of tenants and

[1]We focus on flow setup on host servers (outside VMs); connection establishment within VMs (*e.g.*, TCP handshakes) is outside this paper's scope.

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/2026/08
https://doi.org/10.1145/3789240.3829131

CPU cores on a single server grows, the slow-path flow setup increasingly becomes a bottleneck. As a leading cloud provider, we have adopted a standard Sep-Path design for years, and our DPU's slow path sustains up to 160K connections per second (CPS) for new-flow setup. Nevertheless, it still falls short of customer demands, especially for services that process large numbers of short-lived connections, such as instant messaging and e-commerce promotional events. For example, in our cloud, one such large customer requires >2K CPS per CPU core for new connections. Given that two mainstream specs of our servers are equipped with >350 and >500 CPU cores, respectively, it calls for >700K CPS (350 × 2K) or even >1M CPS (500 × 2K) per server—well beyond what today's slow path can deliver.

Accelerating new-flow setup on host servers in a large cloud is challenging due to the host servers' functional complexity and scale. First, the tables used for a host server's new-flow setup are often much larger than those used in gateways [34, 45, 47, 69] or at the cloud edge [51, 67], because a single host server multiplexes many VMs and flows, and some tables contain rules as fine-grained as the five-tuple. Beyond the size, some tables also require stateful lookups (defined in §2.1). Second, host servers must continuously evolve the table workflow of new flows to meet diverse tenant requirements. For example, one large customer recently required an additional filtering stage with custom rules on top of traditional ACLs and SGs. Third, a large datacenter typically operates >100K host servers, so any acceleration hardware must remain cost-effective at scale. As a result, traditional designs of host-server networking handle new-flow setup on the slow path. Overall, the scale and feature diversity of host servers make *flexibility* and *cost* first-class concerns—alongside *performance*—for the new-flow setup.

To address these issues, we propose *XFir*, the first hardware-accelerated system for new-flow setup (in the slow path) on cloud host servers. XFir achieves high performance (>776K CPS) and high flexibility for constructing new flows, meeting our customers' demands, and is cost-effective to deploy in a hyperscale cloud. At the heart of XFir is a software-hardware co-design built on our next-generation DPU, *Xline*. Xline integrates a general-purpose CPU, a fixed-function ASIC, and a unique accelerator named *Cloud Network co-Processor (CNP)*, which combines a lightweight programmable pipeline with a RISC-V coprocessor. Our key insight is that the CNP is flexible enough to implement the entire slow-path logic for new flows, while achieving substantially higher throughput than CPU-based implementations. Such a co-design is non-trivial, and XFir incorporates three core design components as follows.

First, XFir redesigns the datapath for new flows. Our next-gen DPU introduces an architecture integrating ASIC and CNP, fundamentally different from prior DPUs integrating FPGAs [1] or programmable ASICs [3]. As a result, the conventional software-based slow path design does not directly map to this new architecture and cannot gain the desired performance enhancement. Thus, we redesign the datapath to match the DPU's architecture. Concretely, we migrate the slow-path logic from the DPU's CPU to the CNP, as shown in Figure 2. To fully exploit the CNP's processing units, we elaborately restructure the tables for new-flow setup and lay them out to match the on-chip resource strengths and constraints, enabling the entire flow-setup logic to run efficiently on the CNP.

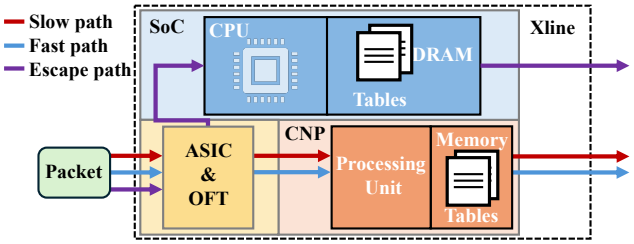


Figure 2: XFir architecture.

Second, XFir optimizes table lookup performance. Processing each new flow involves multiple table lookups, including Longest Prefix Match (LPM) queries over routing, ACL, and SG rules. Since these lookups sit on the critical path, improving their efficiency is essential to accelerating new-flow setup for high CPS. Traditional trie-based implementation in the software plane is usually considered efficient, but it still falls short of our CPS requirement. Therefore, based on our observation that rule prefixes exhibit a highly skewed prefix length distribution, XFir optimizes the table lookup algorithm by combining hashing with tries to reduce memory accesses of hot prefix lengths.

Finally, XFir incorporates the CPU–CNP collaboration in the DPU. Although the CNP offers substantial performance gains for new-flow setup, the DPU's CPU (*i.e.*, the traditional software plane) still provides valuable compute resources that should not be neglected. XFir therefore adopts the CPU to complement the CNP, improving both reliability and throughput. Specifically, we design three core mechanisms: (1) CPU interposition on the offload path to provide observability for monitoring, logging, and debugging; (2) throughput aggregation by processing additional flow setup on the CPU in parallel with the CNP; (3) an escape path that preserves availability when the hardware datapath experiences failures. Together, these mechanisms of CPU–CNP collaboration facilitate XFir's high performance and robust operation.

In summary, the contributions of this paper include:

- We propose XFir, the first hardware-accelerated system for new-flow setup on cloud host servers. In addition to high performance, XFir is flexible for service iteration and is cost-effective for per-host deployment.
- As an early adopter of DPUs with integrated coprocessors, we demonstrate how to leverage this new architecture to solve a long-standing problem in cloud networking, highlighting its significant untapped potential.
- We evaluate XFir comprehensively and compare it against the SOTA FPGA-based DPU solution, Fornax [68]. Our results show that XFir achieves >776K CPS (4.8× that of Fornax), and it reduces the median slow-path latency within the DPU to 11.7μs (69.2% lower than Fornax).

This work **does not** raise any ethical issues.

2 Background and Motivation

2.1 Host Servers and New-Flow Setup

A major service offered by cloud providers is Virtual Machines (VMs), which run on physical machines, and each machine may host one or more VMs. In this paper, we refer to these physical machines as "*host servers*". In practice, a large cloud provider can

operate over 100K host servers in a single datacenter and more than 1M overall.

When an application inside a VM receives/initiates a network flow from/to another endpoint (either within the cloud or on the public Internet), the setup process involves not only the in-VM protocol steps but also a sequence of complex network functions on the host server. The server should lookup the routing table to encapsulate packets with headers for the cloud underlay or for VM delivery. It also enforces tenant- and provider-defined policies by checking the flow against Access Control List (ACL) and Security Group (SG) rules. The key difference of ACLs and SGs is that ACLs are *stateless blacklists*, whereas SGs are *stateful whitelists*.[2]

Generally, we abstract the new-flow setup on a host server as a series of table lookups, including tables for routing, ACL and SG at different granularities, *etc.* In this paper, we focus on this *host-server component of flow setup* and regard the other procedures (*e.g.*, in-VM steps) as orthogonal.

In practice, as application demands evolve, tenants can update these tables by editing entries through cloud APIs. Furthermore, as a cloud provider, we must continuously evolve the processing logic of flow setup. For example, we recently added a filtering stage beyond ACLs and SGs to meet a large customer's requirements; we also restructured table entries to improve performance. Hence, preserving *development flexibility* in new-flow setup, rather than relying on fixed logic, is critical to the customer experience.

2.2 Sep-Path in Hardware Acceleration

The past decades have witnessed the growth of cloud applications and their network traffic. To improve the throughput and latency of cloud networks, providers have continuously explored hardware-assisted networking using devices such as SmartNICs/DPUs, FPGAs, programmable switches, and ASICs. To effectively exploit these accelerators, developers commonly adopt the *Sep-Path* paradigm, which splits packet processing into a programmable software plane and a high-performance hardware plane, and they co-design the two planes to balance flexibility and performance.

The Sep-Path design separates packet processing into a fast path and a slow path. In general, the fast path handles packets that require only lightweight table lookups and pre-defined processing, whereas the slow path handles packets that trigger more complex operations. Accordingly, the fast path is implemented in specialized hardware that offers high performance but limited programmability (*e.g.*, programmable switches or FPGAs). In contrast, the slow path runs in the software plane (*e.g.*, the host's CPU or the DPU's CPU), providing full programmability but lower performance.

At a high level, Sep-Path keeps complete match-action rules in the slow path and selectively offloads compact per-flow entries to the fast path, allowing offloaded flows to be processed at line rate while fast-path misses are redirected to the slow path. This paradigm is widely used in cloud edges [36], gateways [34, 45, 47, 64, 69], and host servers [68]. Specifically on a host server, the slow path handles new-flow setup by looking up tables, consolidating the results into a per-flow entry, and offloading it to the fast path for subsequent packets. Thus, we use "***slow path***", "***new-flow setup***", and "***flow setup***" interchangeably in this context.

[2]The term "*stateful*" means the reverse direction automatically inherits the permit/deny decision of the rule matched in the forward direction. For example, the provider may use ACLs to block VMs from reaching certain internal destinations; a tenant may use SGs to allow inbound Internet traffic only to specific VM ports (and thus automatically permit the corresponding return traffic to the Internet endpoint)

2.3 Slow-Path Bottleneck on Host Servers

In essence, the Sep-Path design leaves the expensive operations in the slow path and does not inherently improve their performance over an all-software datapath. Consequently, as host-server specs scale up, the slow path increasingly becomes the bottleneck for networking.

As a hyperscale cloud provider, we have seen mainstream host-server specs rapidly scale up in compute capacity. In recent years, our typical host servers have grown to >350 and >500 CPU cores (counting hyper-threading), with >800 cores expected soon. Moreover, the attached SmartNIC/DPU bandwidth has increased from 200 Gbps to 400 Gbps, with 800 Gbps on the horizon. However, saturating this scale requires a correspondingly high rate of new-flow setup, especially for applications dominated by a huge number of short-lived ("mice") flows. For instance, one of our largest customers requires >2K new connections per second (CPS) per CPU core, which corresponds to >700K CPS on a 350-core server and >1M CPS on a 500-core server. In contrast, our previous-generation DPU [68], equipped with an on-board CPU with an FPGA accelerator, achieves up to 160K CPS for flow setup in the slow path. This widening gap shows that increasing host compute density makes new-flow setup a growing bottleneck, and it is not automatically accelerated by existing networking hardware.

Hardware acceleration for new-flow setup in host servers is challenging. Unlike gateways [34, 45, 47, 69], peering edges [36, 51, 67], *etc*, the host server's slow path requires substantially more complex table lookups. As discussed in §2.1, these tables serve diverse purposes and may be stateful or stateless, and either provider-defined or tenant-defined. Moreover, host-server tables must be flexible enough to accommodate evolving tenant demands, and the massive scale of host servers constrains per-host hardware investment. As shown in the next section, these constraints narrow the design space for addressing the bottleneck.

2.4 Options to Address the Bottleneck

Running flow setup on a server CPU. A straightforward idea to increase new-flow setup rate is to move the processing from a DPU's CPU to the server's CPU, which is more powerful and offers substantially more cores. Prior work adopts this approach for cloud gateways, achieving around 1.8M CPS with 96 CPU cores [69]. However, unlike gateways deployed on dedicated machines without tenant workloads, host servers must reserve CPU resources for hosting VMs and applications. As a result, CPU resources on host servers are scarce and better allocated to customer workloads rather than slow-path processing.

Attaching multiple DPUs to a server. Another intuitive idea is to adopt multiple DPUs to enlarge the slow path capacity so that the flow setup can be processed in parallel. Prior studies adopt a similar idea for gateways by scaling capacity with multiple accelerators (*e.g.*, 4 or 8 FPGAs) [34, 45, 69]. However, a large datacenter typically contains far more host servers than gateways (>100K vs <1K). This scale difference makes hardware cost a major concern in our design:

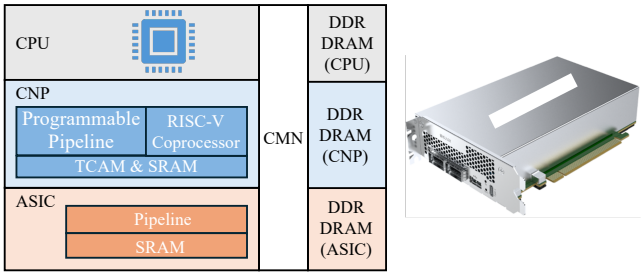


Figure 3: Xline Architecture

doubling or tripling the number of DPUs attached to each server would incur prohibitive financial cost.

Implementing the slow path in an FPGA. Since our previous-generation DPU for host servers integrates a CPU and an FPGA, we initially attempted to migrate the slow path from the CPU to the FPGA. However, we found that an FPGA cannot support the slow path logic because accommodating the necessary tables and control logic would require substantially more resources than the FPGA fabric can hold.

Implementing the slow path in a programmable ASIC. As slow path operations can be abstracted as a pipeline of match-action table lookups (§2.1), we also attempted to implement the slow path on programmable ASICs (*e.g.*, Intel Tofino [22] and AMD Pensando [3]) using P4. However, we found that the slow path does not fit on a programmable ASIC for two reasons: (1) the required tables are too large to fit in the ASIC's limited on-chip memory, even with table merging and folding techniques [45, 47]; and (2) P4's programmability is insufficient to support the level of flexibility needed for continuous feature development (§2.1).

Leveraging existing DPUs. Existing DPU designs do not fully address the new-flow setup bottleneck. NVIDIA BlueField [43] reports high Packets Per Second (***PPS***) by connection tracking [8]. However, this high PPS does not directly translate into high Connections Per Second (***CPS***) for new-flow setup: the connection tracking accelerates packets in *established* connections (the fast path) [44], whereas the slow path involves multiple rounds of processing and table lookups across several packets for each new flow/connection. Moreover, Triton [30] explores a design that departs from the conventional Sep-Path model, but it still handles stateful slow-path operations in a DPU's software plane (see Figure 3 in [30]).

In summary, existing hardware architectures cannot address the performance bottleneck of slow-path new-flow setup on host servers at a practical cost while preserving sufficient flexibility. Instead, our new-generation DPUs, customized for cloud networking, offer a promising alternative.

2.5 Architecture of Our New DPU: Xline

In 2025, we began deploying our next-generation DPU named "*Xline*" at scale across our datacenters. Xline is customized for cloud networking, aiming to provide greater programmability than existing hardware as well as performance acceleration. We note that this paper focuses on the system design of effectively leveraging Xline to accelerate new-flow setup, rather than on the design of Xline itself; nevertheless, for completeness, we briefly introduce Xline in this section.

As shown in Figure 3, Xline integrates three heterogeneous processing components:

- **A 6-core 2.4GHz Arm64 CPU** that provides the same functionality as traditional SmartNIC/DPU. This software plane includes L1, L2, L3 caches and runs Linux, offering full programmability in C/C++.
- **A fixed-function ASIC** that accelerates common packet-processing tasks. It is organized as a pipeline of fixed network functions, such as protocol parsing and Offloaded Flow Table (OFT) lookups, and it exposes no programmability. The ASIC pipeline integrates 2.34 MiB of on-chip SRAM.
- **A Cloud Network co-Processor (CNP)** which is a new component beyond conventional DPUs. The CNP consists of a sequential pipeline followed by a RISC-V processor in both TX and RX. The pipeline provides match-action units with three stages on RX and five stages on TX. The RISC-V processor comprises 32 physical cores with 1GHz; compared to the CPUs in servers and DPU's software plane, these cores are more energy-efficient. The CNP includes a 16K×160-bit TCAM and 2.5 MiB of on-chip SRAM shared by all pipeline stages and RISC-V cores. The pipeline and RISC-V processor are integrated on the same die and directly operate on packet metadata (*e.g.*, parsed headers) in the shared SRAM, avoiding copies between the two components. Although operating on the same on-chip memory, the two components expose different programming interfaces: developers program the pipeline in a P4-like language, whereas the RISC-V cores are programmed in C with inline assembly to explicitly control instructions and memory accesses.

We note that BlueField-3 also integrates RISC-V cores in its Data-Path Accelerator (DPA) subsystem [42]. However, DPA is a software-programmable plane, whereas CNP is a hardware plane: (1) Unlike DPA, CNP's pipeline and RISC-V cores are on-die and remove the OS, DMA, L2/L3 caches, and L1 data cache, retaining an L1-class on-die instruction memory. (2) DPA uses standard RISC-V instructions, while CNP extends the Instruction Set Architecture (ISA) so that common network operations (*e.g.*, hash-table lookup and packet encap/decap) can complete in a single instruction.

Xline also integrates off-chip memory, a DDR DRAM of 16 GiB that serves as a union memory shared among the CPU, ASIC, and CNP, connected via CMN. The DRAM is logically partitioned into three regions for three components, but the CPU and the CNP can be programmed to access any region via physical addresses.

Overall, Xline extends the traditional DPU design with a CNP, a hardware plane that occupies a middle ground between the CPU and the ASIC: it is more programmable than an ASIC and faster than a DPU's CPU, though less flexible than a CPU and less efficient than an ASIC. Improving flow-setup performance based on this new DPU architecture is non-trivial given that we are among the first to explore this design space.

2.6 Design Goals

Given the status quo above, we aim to address the bottleneck of new-flow setup with the following goals:

- **High performance:** Achieve new-flow setup rate that keeps pace with rapidly scaling server specs and NIC bandwidth. In

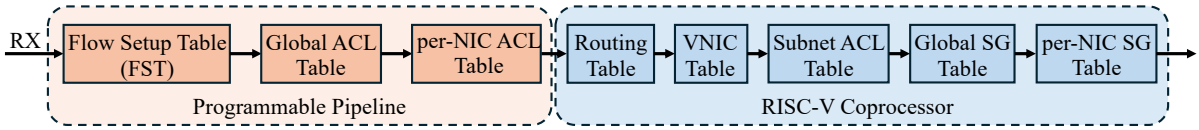


Figure 4: Table layout and workflow.

particular, our target is to exceed 700K CPS on a single host server, while keeping per-flow latency low.

- **High flexibility:** Preserve the ability to continuously evolve host-side networking functionality. The design must support frequent updates to routing and policy processing (§2.1) without requiring redesigns of fixed-function hardware or disruptive re-deployments.
- **Cost-effectiveness:** Keep the deployment economically viable at hyperscale. The solution should require only modest per-host hardware investment and remain sustainable under long-term growth and evolution.

3 XFir Design

3.1 Overview

Figure 2 illustrates XFir's datapath in the host server's DPU. The datapath is still split into a slow path and a fast path, although both traverse the ASIC and the CNP. The two paths differ in the processing they perform. Concretely, the ASIC maintains an *Offloaded Flow Table (OFT)* in DRAM. When the initial packet of a new flow arrives at the DPU, the ASIC misses in the OFT and redirects the packet to the CNP. The CNP then sets up the flow by executing a series of complex table lookups. A flow setup procedure may span multiple packets (*e.g.*, 3 packets in TCP), which are handled similarly and may trigger the same class of lookups. Once the table lookups are completed, the CNP assembles the lookup results into a compact per-flow entry and offloads it to the OFT, allowing the ASIC to efficiently retrieve the required information for the subsequent flow packets. In the fast path, the ASIC still forwards flow packets to the CNP, but the CNP performs only lightweight processing (*e.g.*, pass-through forwarding or a simple table lookup), keeping the fast-path latency low, as shown in our evaluation (§4.3). We route the fast path through the CNP to preserve flexibility for future feature development. Overall, compared to the traditional Sep-Path design, the key difference in XFir is that it runs the slow path in the CNP instead of in the DPU's CPU.

3.2 Datapath and Table Layout

Figure 4 illustrates the table-lookup workflow for ingress flow setup. We omit the egress direction because it follows a similar pattern. The workflow includes: (1) Flow Setup Table (FST), which reduces the following table lookups as described later; (2) the routing table, which maps the destination to its VNIC ID and MAC address for ingress traffic, or to the underlay IP addresses for egress traffic; (3) the VNIC table, which maps a VNIC ID to its associated subnet; and (4) the policy tables, including ACL and SG tables at multiple granularities (*e.g.*, global, per-subnet, and per-NIC).

To achieve reliably high performance, XFir incorporates several techniques in its table designs. For simplicity, we use an ingress TCP flow as a running example in the discussion below and defer UDP and ICMP to later in this section.

FST design. Since table lookups are time-consuming, we design FST to bypass certain expensive lookups. The idea is to reuse the first packet's lookup results for the subsequent packets. Specifically, after the first packet of a new flow completes the workflow in Figure 4 and is permitted by the ACL and SG rules, XFir inserts an FST entry that records the routing information. Hence, subsequent packets during the new-flow setup can then query FST to obtain this information directly, bypassing the routing, ACL, or SG tables. Since an FST entry is created only for permitted flows, it does not explicitly store the permit/deny decision of ACL and SG checks. If a packet misses in FST, it follows the full workflow by querying the tables. Once the flow setup completes, the FST entry is transformed and offloaded to fast path OFT.

Aging and security guarantee. Usually, a FST's entry is removed once the corresponding new-flow setup completes. However, abnormal flows may not complete observably. For example, a client may fail after sending a TCP SYN to the VM, or an attacker may attempt to saturate FST by SYN flooding. We address these concerns by three mechanisms: (1) XFir forcefully deletes any FST entry that has remained for more than 120 seconds. This is similar to the aging mechanism used in prior work [69]. (2) XFir monitors flow-setup rates at multiple granularities (*e.g.*, VPC and VM) and rate-limits anomalous flow-setup activity. (3) Our cloud's DDoS protection system will detect and mitigate the flooding attacks. We omit the details as they are beyond the scope of this paper.

Version control. During flow setup, ACL or SG entries related to the flow may be updated by customers, potentially causing inconsistent decisions across different initial packets (*e.g.*, TCP's three handshake packets). This seems very rare because new-flow setup typically completes much faster than customer operations. However, for popular applications that continuously serve a large number of requests, such inconsistencies can occur more often when customers update ACL or SG rules.

To avoid this issue, we introduce a version control mechanism to the FST and VNIC table. Specifically, the VNIC table maintains a version number for each VNIC, and every change to routing, ACL or SG rules associated with the VNIC or its prefix will increase the version by one. When an FST entry is created, it will also include the current version number in the VNIC table. Therefore, the subsequent establishment packet (*e.g.*, TCP's SYNACK and ACK) will compare the version number obtained from FST with the current one in the VNIC table. If the versions do not match, the packet must query the subsequent ACL and SG tables across different granularities; otherwise, it skips them.

Table layout in the pipeline and the coprocessor. The CNP integrates a pipeline and a RISC-V coprocessor, which offer different features. Although the pipeline is fast, it cannot host the entire workflow due to limited stages and on-chip resources, and it provides insufficient programmability for stateful tables such as SG. Conversely, handling all lookups on the coprocessor cannot achieve the best performance.

Therefore, XFir adopts a hybrid table layout. As shown in Figure 4, we assign FST, the global ACL, and the per-NIC ACL to the pipeline, and keep the remaining tables on the coprocessor. This layout balances performance gains, on-chip memory constraints, iteration agility, and functional requirements as follows.

Concretely, the FST is the first lookup stage because every packet consults it. To accommodate the number of concurrent flow setups under high CPS, the FST has a large size and is stored in DRAM. Meanwhile, the global ACL and per-NIC ACL are relatively stable: the global ACL blocks a small set of internal addresses in the cloud, and the per-NIC ACL is enabled only for selected customers. These tables are small enough to fit in the CNP's TCAM for low-latency access. Moreover, performing these ACL checks early in the pipeline filters invalid packets upfront, reducing unnecessary lookups in later tables.

As for the other tables, some require higher flexibility for implementation or iteration. For example, SGs are stateful, meaning that once a packet is permitted in one direction, the packet in the reverse direction is also permitted (§2.1). Moreover, some tables impose ordering constraints due to their functionality. For instance, the VNIC table depends on the routing-table result; the ACL and SG tables after the VNIC can be bypassed via the FST and version control mechanism. As a consequence, XFir handles these tables in the coprocessor. In addition, these tables are large because their entries have long lifetimes tied to VMs on the host server. Therefore, we store them in DRAM.

Putting it all together. Given our redesigned table layout, we now present the end-to-end flow-setup workflow in CNP.

(1) When the first packet (SYN) of a TCP flow arrives, it misses in the FST and therefore traverses the full workflow. It spends 0.5~2μs in the pipeline and 4~10μs in the coprocessor. If the SYN packet is permitted after the ACL and SG checks, XFir will create and inserts an FST entry.
(2) Since the FST is shared between TX and RX, the SYNACK packet will find a matching FST entry, and the version in the entry also matches the one in the VNIC table. Thus, it bypasses the rest of ACL and SG tables. Note that it still goes through the global ACL and the per-NIC ACL in the CNP pipeline, but their lookups are extremely fast (<1μs) due to TCAM.
(3) When the third packet (ACK) arrives, it follows a similar procedure as SYNACK without lookup in the ACL and SG tables.
(4) Then the TCP flow is considered established. Thus, the corresponding FST entry is offloaded to the OFT in the fast path to accelerate the subsequent packets of the flow.

UDP. A naive approach for a new UDP flow is to offload it to the fast path as soon as the first packet completes the slow path. However, this can waste fast-path resources when senders create small UDP flows with only a single packet. To reduce such unnecessary offloading, XFir delays offloading a UDP flow until it has received three packets. Specifically, the first packet creates an FST entry, and the second and third packets in the same direction are still handled by the slow path but benefit from FST acceleration. After that, the FST entry is offloaded to the fast path. Overall, XFir uses three packets as a heuristic threshold that indicates a UDP flow is likely meaningful for offloading.

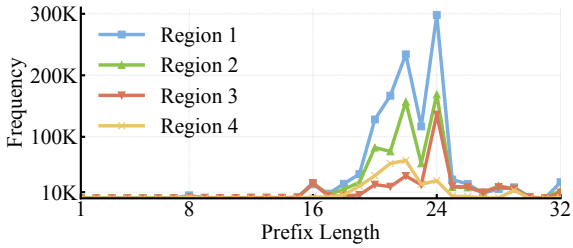


Figure 5: Frequency distribution of prefix lengths.

ICMP. For a new ICMP flow, the first request packet (sequence number 1) is handled by the slow path and creates an FST entry. The corresponding reply then finds a matching FST entry, after which XFir offloads the entry to the fast path to handle subsequent packets. In other words, XFir offloads an ICMP flow after completing the first round trip with sequence number 1 in the slow path.

As a summary, the new datapath in the CNP speeds up the new-flow setup and also provides desirable flexibility for continuous development.

3.3 LPM Lookup Optimization

In XFir, most of the slow-path tables (routing, ACLs, and SGs) must frequently perform Longest Prefix Match (LPM) to lookup rules based on a packet's IP address, which lies on the critical path of new-flow setup. The traditional lookup algorithm based on trie or Patricia trie [39] is actually not hardware-friendly because it requires multiple non-contiguous memory accesses, especially when stored prefixes create branches at different depths.

A common optimization is to store rules in TCAM, which naturally supports prefix matching. However, TCAM is significantly more expensive than a DRAM or SRAM [61], and thus its capacity is typically very limited in hardware. For example, Xline is equipped with 16 GiB DRAM but only 16K×160bit in the CNP. As a result, XFir reserves TCAM for the ACL tables in the CNP pipeline [3], stores large tables in DRAM (§3.2), and optimizes LPM for tables processed by the RISC-V cores to reduce DRAM accesses.

Researchers have extensively explored LPM optimization. Drawing on several ideas from the literature, including distribution-specific optimization [11, 18, 59], prefix-length division [12, 13, 58, 65, 66], and hybrid trie designs with auxiliary data structures [4, 6, 15, 58], XFir adopts an LPM algorithm that balances performance, memory consumption, and implementation complexity for cloud host servers. We compare our algorithms with existing ones in §6.

Our algorithm is based on an observation in cloud networks: although the prefixes span the full range of lengths, the lengths are far from uniformly distributed. As an example, we collect IPv4 prefix lengths from host servers' subnet routing tables in four of our cloud regions, and the distribution is shown in Figure 5. Although the total number of prefixes varies across regions, they share a similar pattern in the distribution: Most prefixes have lengths between 17 and 24 bits—over 80% for Regions 1–3 and over 65% for Region 4. Actually, we also observe this pattern for prefixes in ACL and SG tables. Based on this observation, our lookup algorithm combines

[3] FST in the pipeline uses exact six-tuple match (VPC ID + traditional five-tuple) instead of LPM, and it is stored in DRAM

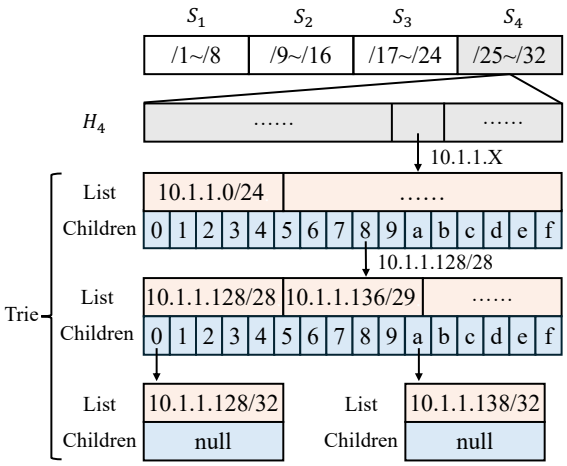


Figure 6: XFir's hash+trie data structure.

hashing with tries, to accelerate the lookups for longer prefixes. For simplicity, we present the algorithm for IPv4 as follows.

Data structure. As illustrated in Figure 6, given a collection of prefixes with varying lengths, we divide them into four subsets S_i ($1 \le i \le 4$), where S_i contains prefixes whose lengths fall into $[8(i-1)+1,\ 8i]$. For each S_i, we build a hash table H_i of size M_i, and each prefix is mapped to a slot by hashing its high-order $8i$ bits. Hence, each slot aggregates prefixes in the same length range $[8(i-1)+1,\ 8i]$ that share the same $8i$-bit key (if no hash collision).

We then build a trie for each hash slot to store these prefixes and associated entries. The trie has radix 16: each node maintains 16 child pointers indexed by the next 4 high-order bits. To insert a prefix, we traverse the trie from the root in 4-bit strides until the prefix has less than four bits. Then we store the prefix and the associated entry at the corresponding node. A node may contain multiple prefixes (up to four) that have different lengths but share the same path up to the current 4-bit stride (*e.g.*, 10.1.1.128/28 and 10.1.1.136/29). Thus, each node maintains a list of up to four slots, each corresponding to a prefix. To save memory, each slot stores only a validity field rather than the prefix itself; the exact prefix can be reconstructed from the traversal path during lookup. Prior work [15] proposes to use a single stride-width bitmap in each node to encode prefix validity and further reduce memory consumption and accelerate lookup. However, this design cannot be directly applied to XFir due to the requirements of priority-aware LPM, as described later in this section.

Lookup algorithm. Given an IP address, our goal is to find its LPM in the hash+trie structure. The algorithm searches the subsets in descending order, starting from S_4 and then S_3, S_2, and S_1. The first match found in this order is the LPM because S_i with higher i stores longer prefixes. For S_i, the algorithm hashes the high-order $8i$ bits of the address and thus obtains the trie root in the hash slot of H_i. It then traverses the trie using the remaining address bits. During traversal, the algorithm selects the longest valid prefix at each visited node and records the most recent match along the path. The traversal stops when the next child pointer is null, and the last recorded entry is returned as the LPM in S_i.

The lookup falls back to S_{i-1} when no match is found in S_i. This can occur in two situations: (1) the hash slot is empty; (2) none of the prefix lists at the visited nodes contains a matched prefix. In both cases, the algorithm continues to find the matched prefix in S_{i-1}. If it misses in all subsets, the algorithm reports no match.

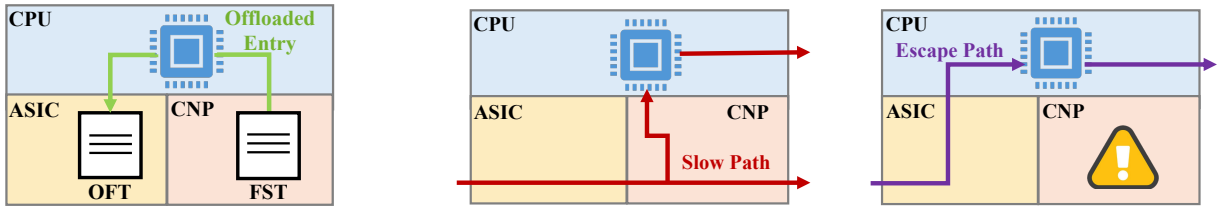
Analysis. Our design improves LPM lookup efficiency by exploiting two properties. Firstly, it leverages the skewed prefix-length distribution and searches from longer to shorter ranges, so lookups typically succeed in S_4 or S_3 and avoid traversing S_2 and S_1 for shorter prefixes. Secondly, we adopt a radix-16 trie (4-bit stride) in a hash slot. Compared to the 1-bit stride, the 4-bit stride significantly reduces trie height, and thus reduces the number of memory accesses required per lookup.

Given a collection of N prefixes, the total space for all tries is $O(N)$. The total space for all hash tables is $O(\sum_i M_i) = O(\frac{N}{\alpha})$, where α is the load factor. Therefore, the overall space complexity of the data structure is $O(N)$. Let m denote the number of subsets (S_i), r the trie stride in bits, and l the bit length of an IP address. A worst-case lookup checks all m subsets and traverses one trie in each subset, yielding a time complexity of $O(m)+O(m\cdot\frac{l}{mr}) = O(m+\frac{l}{r})$. For IPv4 ($l=32$), we use $m=4$ and $r=4$; for IPv6 ($l=128$), we use $m=8$ and $r=4$. Although the space and time complexity is close to a traditional trie, the number of memory accesses is significantly reduced for LPM with common longer prefix lengths (see Table 1 in §4.1 for evaluation results).

The choice of subset count m reflects a tradeoff in lookup performance. A larger m creates finer-grained subsets and reduces trie height, but it also increases the number of hash probes across subsets when matching shorter prefixes. In contrast, a smaller m reduces the number of hash probes, but may not fit the prefix-length distribution in Figure 5 well, providing less benefit for common longer prefixes. We choose $m=4$ as a balance between these two cases. Furthermore, one could improve performance by using variable subset ranges tailored to the prefix-length distribution. However, this would increase design and implementation complexity, and the performance benefit may degrade when the prefix-length distribution changes. We leave this exploration to future work.

Adding VPC ID to LPM. In a cloud, each VM's NIC is assigned to a Virtual Private Cloud (VPC), within which a tenant can allocate subnets and manage ACL, SG, and peering rules. Thus, the routing, ACL and SG rules are associated with both VPC ID and IP prefixes, and the LPM must be performed within the same VPC ID. Accordingly, XFir uses the VPC ID along with the IP address as the lookup key. In the hash+trie design, the VPC ID and the high-order address bits are used for hashing, while trie traversal depends only on the IP address bits. To resolve hash collisions, XFir builds a separate trie for each distinct combination of VPC ID and high-order address bits mapped to the same hash slot. Incorporating VPC IDs does not affect the time or space complexity which depends on the total number of prefixes instead of the number of VPCs.

Priority-aware LPM. In cloud networks, the same prefix may appear in the same VPC under different scenarios. For example, a customer-defined subnet prefix may coincide with a prefix automatically assigned to a public but VPC-specific service, such as object storage (*e.g.*, Amazon S3 [5]). Such coincidences are hard to avoid in practice because the service may be enabled either before or after the customer creates the subnet. To allow these prefixes to coexist, a cloud allows customers to assign priorities to prefixes.



(a) Control and observability of the offload path. **(b) CPU–CNP parallel flow serving.** **(c) Escape path through the CPU.**

Figure 7: Three mechanisms of CPU–CNP collaboration.

For example, when a service prefix coexists with a subnet prefix on the same host, the service prefix with a higher priority will be selected for routing. Thus, during LPM, if multiple prefixes with the same longest length match an address, XFir should return the one with the highest priority.

To support such a priority-aware LPM, each slot in a trie node stores an additional priority bitmap. Each bit indicates whether a prefix with the corresponding priority exists, with higher-order bits representing higher priorities. After identifying the longest matching prefix, the lookup algorithm checks this bitmap to select the highest-priority match and returns the priority along with the matched entry. Furthermore, when priority is provided as an input parameter, this design also supports LPM queries within the specified priority: during traversal, the lookup algorithm checks whether the priority bits are set in the slots of visited nodes.

3.4 Collaboration of CPU and CNP

XFir uses the CNP to accelerate new-flow setup and the ASIC to accelerate subsequent packets at high speed. Nevertheless, the DPU's CPU, as a processor with full programmability, remains essential for both operational reliability and further throughput enhancement. In XFir, the CPU actively collaborates with the CNP to handle three major tasks as shown in Figure 7 and described below.

Controlling and observing the offload path (Figure 7a). Ensuring reliable hardware acceleration requires both control and observability over the offload path, especially for a new architecture with a redesigned datapath like Xline. In XFir, the CPU mediates the offload path and records offload events for monitoring and debugging. To be specific, the CPU continuously polls the FST in DRAM via physical addresses (§2.5) to promptly detect finished flow setup. Once an entry is marked as completed, the CPU logs the entire entry, translates it into the format accepted by the ASIC, and installs it to the OFT in the fast path via the offload channel. Then the CPU removes the entry from the FST. By interposing on the offload path, the CPU controls and observes which flows are offloaded and how the offload proceeds.

One may consider offloading flow entries directly from the CNP to the ASIC to further reduce offload latency. XFir instead chooses CPU interposition for the following reasons. (1) It allows us to reuse the mature monitoring, logging, and debugging infrastructure available on the CPUs of traditional DPUs. As an early adopter of Xline-like DPUs, we prioritize reliability and rapid deployment: reusing existing tooling simplifies deployment, validation, and debugging during rollouts. (2) CPU polling and the offload (CPU–ASIC) channels have ample capacity and low latency, so the CPU interposition adds only a very small delay (<10μs) and does not become a throughput bottleneck (>2Mpps). (3) The offload path is used once per flow, further reducing the performance impact.

Serving flow setup in parallel (Figure 7b). While the CNP substantially improves the new-flow setup rate, the software plane (the DPU's CPU) still provides non-trivial capacity that should not be wasted. As host servers' demands continue to grow (*e.g.*, a 500-core server may demand >1M CPS; see §2.3), leveraging both the CNP and the CPU in parallel becomes a practical way to scale total CPS.

To achieve such collaboration, we devise a table-splitting mechanism that separates new flows by prefixes, enabling the CPU and the CNP to perform table lookups on disjoint traffic simultaneously. Specifically, we split new-flow traffic into two parts: the CNP handles prefixes with high flow frequency, while the CPU handles low-frequency prefixes. We use the global ACL table in Figure 4 as the split point. For prefixes assigned to the CPU, we move the original matched action to the table in the CPU and install a "redirect-to-CPU" action as the default action for packets that do not match the CNP-handled prefixes. Therefore, when a packet with a low-frequency prefix arrives, it misses in the global ACL table and is redirected to the CPU, which then performs the corresponding lookup and continues the remaining table lookups for flow setup. We also split the downstream tables after the global ACL table (see Figure 4) between the CPU and the CNP according to the prefixes and associated VNICs. The CPU stores its portion of table entries in the CPU's DRAM region (see Figure 3). Once a packet is redirected, all subsequent lookups are naturally handled by the CPU. As a result, a new flow may be handled by either the CPU or the CNP during flow setup, but it is never processed by both concurrently, avoiding conflicts or races between the two paths.

Because the FST precedes the global ACL table, both tables must handle the combined traffic before it is split between the CPU and the CNP. As the CNP pipeline runs at line rate, it can perform these lookups at a rate that accommodates the aggregate CPS of both datapaths. Moreover, to preserve the benefit of FST for CPU-handled flows, the CPU directly accesses FST via physical addresses (§2.5) and inserts an FST entry after completing the lookup for the first packet of a new flow.

Overall, by processing new flows of different prefixes in parallel on the CPU and the CNP, XFir can further increase CPS.

Providing the escape path (Figure 7c). Hardware failures are not rare at a large deployment scale. Therefore, the CPU also provides an escape path when the CNP encounters failures. Once the CNP is detected to be unresponsive or enters an error state, the CPU installs a high-priority entry in the OFT in the ASIC to redirect traffic to the CPU, bypassing the CNP. The CPU then takes over the new-flow setup and, if needed, handles subsequent packets as well. This design preserves service availability and maintains

Algorithm	Prefix Length	# DRAM Accesses	Time (μs)
Hash + Trie	/25 ~ /32	≤ 4	≤ 0.44
	/17 ~ /24	≤ 7	≤ 0.74
	/9 ~ /16	≤ 10	≤ 1.04
	/1 ~ /8	≤ 13	≤ 1.33
Trie	/25 ~ /32	26 ~ 33	3.69 ~ 4.88
	/17 ~ /24	18 ~ 25	2.52 ~ 3.55
	/9 ~ /16	10 ~ 17	1.33 ~ 2.37
	/1 ~ /8	2 ~ 9	0.15 ~ 1.18

Table 1: The number of DRAM accesses for trie node retrieval and lookup time for different lengths of matched prefixes.

correct policy enforcement under CNP failures, albeit with reduced throughput compared to the normal CNP-accelerated path.

Failure detection is performed by our active-passive combined hardware monitoring system. We omit its design details because they are beyond the scope of this paper.

4 Evaluation

In this section, we evaluate XFir in terms of optimized LPM lookup time (§4.1), new-flow setup throughput in CPS (§4.2), latency (§4.3), and deployment cost (§4.5). Moreover, we compare XFir against two schemes: SW and Fornax.

- ***SW*** represents a slow path in Xline's software plane: we configure Xline's ASIC to redirect packets to Xline's CPU for slow-path processing, bypassing the CNP.
- ***Fornax*** [68] is a state-of-the-art FPGA-based Sep-Path design for per-host deployment in a cloud. Fornax supports 16M concurrent sessions with policy enforcement in the fast path, and XFir's fast path provides the same capability through cooperation between CNP and ASIC, making the comparison fair.

Testbed. For XFir and SW, we use two host servers in the same rack of our datacenter, each equipped with an Xline DPU whose on-board CPU is a 6-core 2.4GHz Arm64 processor. For Fornax, we use another two same-rack servers, each equipped with our previous-generation DPU, which integrates a 4-core 2.9GHz Intel x64 CPU and an FPGA. The CPUs in both DPU platforms run CentOS. Moreover, in end-to-end experiments, we run a 128-vCPU (AMD64) VM on each host server across all schemes.

4.1 Optimized LPM Lookup Time

We evaluate XFir's optimized LPM lookup algorithms (§3.3) by computing the number of DRAM accesses for trie node retrieval and measuring lookup latency in the CNP's coprocessor. Table 1 reports the results across different matched-prefix lengths.

In Table 1, we group prefix lengths into four ranges following the division in §3.3. Since our hash+trie algorithm starts from the group of longest prefixes (/25~/32), lookups whose matches fall into the first two groups incur the lower cost. In the best case, when the matched prefix length falls in 25~32bits, the lookup requires at most four DRAM accesses and takes 0.44μs. When the matched prefix length is in 17~24bits, the algorithm first checks the group of /25~/32 and then continues to the second group (/17~/24) after a miss. It takes at most seven DRAM accesses, corresponding to 0.74μs. In the worst case, a short prefix requires 13 DRAM accesses and takes 1.33μs.

In contrast, a traditional trie incurs higher latency for longer prefixes because it must traverse more nodes, leading to more DRAM accesses. It takes 4.88μs and 3.69μs for prefix lengths of 32 and 25, respectively. Therefore, since most customer rules in our slow-path tables have prefix lengths between 17 and 24 (See Figure 5), the hash+trie algorithm in XFir typically achieves substantially lower lookup time than a traditional trie.

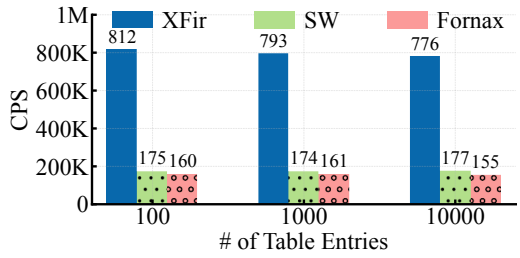


Figure 8: New-flow setup rate (CPS). Numbers above bars are in thousands (K).

4.2 Connections Per Second (CPS)

We measure the throughput of new-flow setup for three schemes (XFir, SW, and Fornax) in terms of CPS. For each scheme, a VM on one host server initiates TCP connections to a VM on another host server as fast as possible. We run each experiment for 20s and compute CPS using the connections established during the middle 10s interval. The CPS is computed by the number of established connections in that interval divided by 10s. The new-flow setup in each scheme involves a series of table lookups to realize desired network functions, so we set up three settings by varying the number of entries in each table (100, 1K, 10K). We report the CPS for all three settings for each scheme. We ensure that network bandwidth is not a bottleneck in these experiments: even at the maximum CPS, the traffic volume is below 10Gbps, far lower than the DPU bandwidth (≥200Gbps) of all schemes.

As shown in Figure 8, XFir achieves the highest CPS among the three schemes, ranging from 776K to 812K. In contrast, SW and Fornax reach only around 175K and 160K CPS, respectively. Thus, XFir achieves 4.4× and 4.8× the CPS of SW and Fornax, respectively, demonstrating the effectiveness of our design based on the new DPU architecture, Xline. SW slightly outperforms Fornax, which is attributed to the two additional CPU cores in Xline compared to Fornax's DPU. Moreover, the number of entries in the tables has little impact on CPS across all three schemes, indicating that their throughput is largely insensitive to table size.

The over 776K CPS in Figure 8 is achieved using XFir's CNP alone, without enabling the CPU–CNP parallel flow-serving mechanism described in §3.4. Since 776K CPS already satisfies the demands of most of our customers, we decide to ship this version of XFir as early as possible and defer the parallel flow-serving implementation to near-term future work. This upgrade can be realized via program update in the CPU and the CNP, without hardware changes. Nevertheless, Figure 8 allows a back-of-the-envelope estimate by adding the XFir CPS and the SW CPS. The upgraded XFir could reach

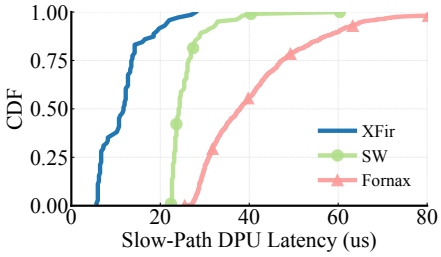


Figure 9: Slow-path DPU latency.

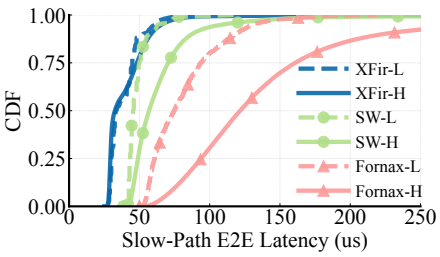


Figure 10: Slow-path E2E latency with light (L) and heavy (H) workload.

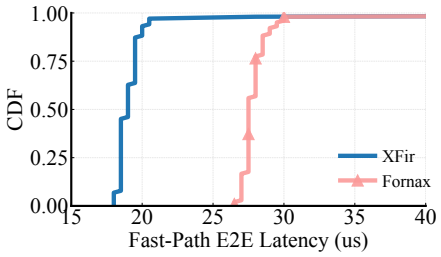


Figure 11: Fast-path E2E latency.

roughly 776K+175K=951K, approaching the ~1M-CPS requirement for a host server with 500 cores discussed in §2.3.

Overall, XFir outperforms the state-of-the-art FPGA-based scheme, Fornax, achieving 4.8× higher CPS for the slow path.

4.3 Latency

In this section, we evaluate the one-way latency of both the slow path (new-flow setup) and the fast path (subsequent traffic) for XFir, SW, and Fornax. We measure two metrics: *DPU latency* and *end-to-end latency*.

- ***DPU latency*** measures the time a packet spends inside the DPU, including computation and internal bus/channel transfer. It reflects DPU efficiency.
- ***End-to-end (E2E) latency*** measures VM-to-VM packet latency between two host servers, covering the path from the sender VM's network stack through the hypervisor to the DPU, and the corresponding path from the receiver-side DPU to the destination VM's network stack. Since the two servers are in the same rack, propagation delay is negligible. This latency includes DPU latency, and it captures tenant-perceived performance.

DPU latency of the *slow* path. Figure 9 plots the CDF of slow-path DPU latency measured using 100 TCP SYN packets. XFir achieves the lowest latency. In contrast, SW and Fornax exhibit a much wider latency spread, with Fornax showing the highest DPU latency. Specifically, the p50 latency of XFir, SW, and Fornax is 11.7μs, 24.3μs, and 38.0μs, respectively.

XFir's performance gains stem from two factors: 1) the CNP's pipeline and coprocessor can process packets and table lookups efficiently. 2) XFir's design fully exploits the Xline architecture through careful redesign of table layout and optimization of lookup algorithms, accelerating the complex table lookups.

E2E latency of the *slow* path. We measure slow-path end-to-end latency by sending multiple TCP SYN packets in parallel from a VM to another VM. We compute the RTT as the time between sending a SYN and receiving the SYNACK, and divide it by two to obtain one-way latency. In practice, DPUs will serve diverse VM and application workloads, which can affect per-packet latency. Therefore, we evaluate all three schemes under two workload settings by varying the SYN sending rate: (1) a *light workload* of 100 CPS; and (2) a heavy workload at each scheme's maximum CPS.

Figure 10 reports the results under light and heavy workloads. In both settings, XFir achieves substantially lower latency than the other two schemes. Specifically, under the light workload, the p50 latency of XFir, SW, and Fornax is 34μs, 46μs, and 75μs; under the heavy workload, the p50 becomes 33μs, 67μs, and 122μs. These results indicate that increasing workload has a smaller impact on XFir's slow-path latency than on SW and Fornax. Tail latency shows an even larger gap: under the heavy workload, Fornax's p99 reaches 6738μs, compared to 87μs and 194μs for XFir and SW, respectively.

DPU latency of the *fast path.* Although XFir's primary contribution is accelerating the slow path, we also measure fast-path DPU latency for completeness. The fast path is not applicable to SW, so we compare XFir and Fornax. This latency is highly stable: XFir remains in 1.0~1.05μs, while Fornax stays within 2.0~2.1μs (figure omitted). The fast-path latency benefit is mainly due to more efficient packet processing in XFir's ASIC than on Fornax's FPGA.

E2E latency of the *fast path.* We measure fast-path end-to-end latency using the same VM-to-VM probing method as in the slow path. We adopt 100 ICMP request/response pairs handled by the fast path to compute the RTT, which is divided by two to obtain one-way latency. The results are plotted in Figure 11. XFir consistently achieves lower fast-path latency than Fornax. Specifically, the p50/p99 latencies are 19μs/28μs for XFir and 27μs/30μs for Fornax. Overall, XFir reduces the median fast-path E2E latency by 8μs (29.6%) compared to Fornax.

4.4 Ablation Study

We conduct an ablation study to evaluate the effectiveness of XFir's table-layout design described in §3.2. We compare XFir against two variants, XFir−V and XFir−VP:

- ***XFir−V*** removes version control from XFir, forcing every packet during new-flow setup to query the ACL and SG tables after the VNIC table in Figure 4.
- ***XFir−VP*** additionally moves the tables processed by the pipeline to the RISC-V cores, on top of removing version control. This represents the case where the table layout is not tailored to the CNP architecture.

We measure the CPS of these variants using the same methodology as in §4.2 and report the results in Figure 12. First, removing version control reduces CPS from 776K~812K (XFir) to 676K~700K (XFir−V), a drop of about 100K. Further moving the pipeline tables to the RISC-V coprocessor lowers CPS to 665K~681K (XFir−VP), an additional drop of about 20K. Hence, splitting tables between the pipeline and the RISC-V coprocessor contributes less to the CPS improvement than version control does, because the tables that remain in RISC-V dominate the overhead of new-flow setup. Overall, both version control and the table-layout design contribute to XFir's CPS improvement.

Moreover, XFir−VP still achieves a high CPS (compared with SW in Figure 8), indicating that most of the CPS improvement comes

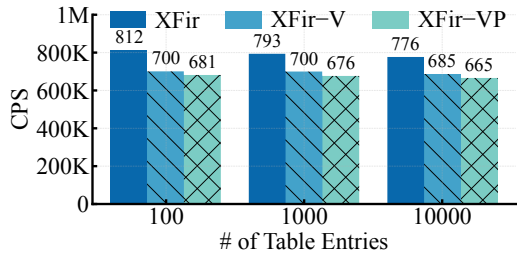


Figure 12: Ablation study of CPS. Numbers above bars are in thousands (K).

from the acceleration provided by the CNP. The CNP's RISC-V coprocessor delivers much higher performance at a lower frequency than the CPU in the software plane, because it provides instructions specialized for common network operations (as described in §2.5) and because we effectively leverage these instructions when implementing the flow-setup logic.

4.5 Deployment Cost

We compare the cost of XFir with several recently proposed hardware-accelerated systems, as summarized in Table 2. Since absolute pricing is proprietary, we mask real prices by reporting relative costs normalized to XFir. For each system, we consider both hardware cost and power cost. We estimate hardware cost by summing the per-component prices quoted by our manufacturer. For power cost, we estimate total energy consumption over three years, matching our typical hardware refresh cycle. We add up the power consumption of all hardware components and multiply it by the electricity price quoted by our provider.

Table 2 presents the normalized cost and the cost per 100Gbps for each system. Tiara [69] and Albatross [34] are gateways deployed on dedicated servers and are provisioned at a scale of only a few hundred per datacenter. Hence, they target high bandwidth/throughput at high cost. Although they can exceed 1M CPS, their design is not applicable to host-server deployment, which involves >10K servers per datacenter (see §2.4). In contrast, Fornax [68], Triton [30], and XFir target per-host deployments with one DPU per server, and they are therefore designed for lower cost at host-level bandwidth. Compared to Fornax and Triton, XFir has a slightly higher total cost but achieves a lower cost per 100Gbps. Moreover, as shown in §4.2, XFir delivers significantly higher new-flow CPS than the FPGA-based approach.

Overall, XFir achieves low cost yet high performance. XFir is highly cost-effective for four reasons: (1) Xline adopts a cost-efficient architecture built around a fixed-function ASIC and a CNP that combines a lightweight pipeline with a low-end coprocessor. (2) Xline extends the RISC-V cores with network-specific instructions, improving both execution and energy efficiency. (3) XFir fully leverages Xline through a carefully designed table layout (§3.2) and optimized lookups (§3.3). (4) We tune the coprocessor code of C and assembly to reduce cycle overhead for packet processing.

5 Discussions

Deployment status and plan. We have deployed the new-generation DPU, Xline, as the primary DPU for host servers in our datacenter since 2025, where it has been used to improve the performance and

System	Hardware Spec	BW (Gbps)	Cost	Cost /100Gbps
Fornax[68]	DPU (4-core CPU +FPGA+DRAM)	2×100	0.92	0.46
Triton[30]	DPU (8-core CPU +FPGA+DRAM)	2×100	1	0.50
Tiara[69]	48-core server CPU +8×FPGA+Tofino +DRAM	8×2×100	12.5	0.78
Albatross [34]	96-core server CPU +4×FPGA+DRAM	4×2×100	5	0.63
XFir	DPU (6-core CPU +Fixed ASIC+CNP +DRAM)	2×200	1	0.25

Table 2: Cost of hardware-accelerated systems.

flexibility of the *fast* path. As for XFir, which targets enhancing *slow* path (new-flow setup), we have been running canary testing for over half a year on about 1K host servers. To ensure safe deployment and high service reliability, we are conducting a staged rollout of XFir.

Generality of XFir's design. Although XFir is tailored to Xline, several design rationales can generalize to software implementations and other hardware platforms, including the FST design, version control, and the LPM algorithm. The CPU–CNP collaboration is specific to Xline, but the same idea may apply to other heterogeneous DPUs, such as enabling CPU–DPA collaboration on BlueField-3. These design rationales can help improve CPS on other hardware or software, but without Xline's hardware architecture, it remains difficult to achieve CPS as high as XFir.

Performance comparison with commercial DPUs. Due to the limited availability of commercial DPUs in our cloud, we cannot directly compare XFir's new-flow setup performance against implementations on other commercial DPUs, such as NVIDIA BlueField-3 [43] and AMD Pensando [3]. Instead, we reason about their expected performance from their hardware architectures and the requirements of new-flow setup.

As discussed in §2.4, new-flow setup relies on a series of large stateful and stateless tables and demands high flexibility. For AMD Pensando, the limited resources of its P4 pipeline make it hard to host the complete new-flow setup logic, which therefore must be handled by its embedded software plane (Arm cores). For NVIDIA BlueField-3, the logic can instead run on the DPA subsystem [42]. However, DPA is a software-programmed RISC-V processor rather than a hardware plane like the CNP (§2.5), so we regard a DPA implementation as a software-plane one. Moreover, recent studies point to two weaknesses of the DPA that suggest it is less suitable for new-flow processing: its low-profile RISC-V cores are much weaker than the Arm cores [71], and its weaker memory subsystem causes performance to degrade significantly once the working set exceeds the cache capacity [10].

Therefore, we consider new-flow setup on these commercial DPUs as a software slow-path solution within the Sep-Path design. We expect their performance to be close to the software baselines (SW and Fornax) evaluated in this paper, and to remain below that of XFir. However, this expectation is a qualitative estimate rather

than a rigorous measurement, and that the actual gap may vary with implementation. We leave a direct experimental comparison on commercial DPUs to future work.

Evaluation scale. Our evaluation was conducted on only two host servers, with one host processing new flows initiated by the other. Nevertheless, we believe this setup approximates the per-host performance in a large-scale incast scenario, where one host receives new flows from many servers or clients. This is because new-flow setup is performed independently for each flow on the receiving DPU, and each flow follows the same table-lookup workflow regardless of which or how many senders initiate the flows.

Toward 2M CPS in the future. As DPU bandwidth and CPU core counts continue to grow, they impose increasing demands on the new-flow setup rate. Our next DPU iteration, Xline-2, targets 2×400 Gbps and deployment on servers with >800 CPU cores. To keep pace with this evolution, we aim to scale XFir2 to 2M CPS by further refining our software-hardware co-design and unleashing its performance potential.

6 Related Work

Hardware acceleration for networking. With the rapid evolution of network hardware, a large body of prior work has explored accelerating server networking via software-hardware co-design. For example, researchers have offloaded components of the network stack or network functions (NFs) to various hardware, such as SmartNICs [16, 25, 28, 38, 48, 52], DPUs [30, 68], FPGAs [29, 30, 34], programmable ASICs [20, 33, 36, 37, 47, 49, 53], and commodity switches [17], *etc.* They also propose combining heterogeneous hardware to accelerate complex NFs (*e.g.*, those requiring per-flow state), such as programmable switches with RDMAs [26, 50] and programmable switches with FPGAs [45, 69].

However, existing approaches do not meet our demands for accelerating flow setup on cloud host servers. Firstly, offloading NFs or parts of the network stack to SmartNICs or DPUs [16, 25, 28, 30, 33, 38, 48, 52, 68] has struggled to achieve the desired CPS for new flow setups due to the limited capability of NFPs or CPUs in SmartNICs/DPUs, as discussed in §2.3. Secondly, several studies adopt the "Sep-Path" paradigm by placing the slow path (flow setup or other complex NFs) in the software plane of the hardware accelerator [37] or of the server [16, 20, 29, 47, 49, 50, 52, 53, 68, 69], which fail to provide high CPS and consume the precious host-server compute resources as discussed in §2.4. Finally, some proposals cannot support per-flow state [25, 26] or target only specific applications [20, 33] or specific NFs [17].

Overall, XFir aims to accelerate *flow setup* on cloud host servers while maintaining high flexibility and low cost—a unique challenge that existing work does not achieve.

Software improvement for networking. Prior work has also explored improving the network stack's performance in software. For example, a series of studies adopt user-space networking [14, 23, 35] and OS/kernel extensions [7, 31] to speed up the entire network stack. Another line of work improves connection dispatch efficiency for applications via kernel optimizations [32] or co-scheduling based on worker status [46]. These studies enhance the software implementation of connection establishment (*i.e.*, protocol handshakes), and they are orthogonal to XFir. In contrast, XFir focuses on the flow setup on multi-tenant host servers (as defined in §2.1). Moreover, XFir can incorporate these techniques to further optimize the host servers' software plane.

LPM algorithm optimization. Researchers have long studied how to improve the time and memory efficiency of LPM. Starting from trie-based algorithms, prior work has proposed techniques such as leaf pushing [6, 11, 58], prefix division and path compression [15, 40], as well as hybrid designs that combine tries with other data structures [4, 6, 15, 58]. We build on these ideas and combine hashing with tries for the XFir setting. The closest approaches to our design are hierarchical hashing [59] and SHIP [58]. Hierarchical hashing [59] uses the same prefix division scheme as ours, but relies on multiple levels of hash tables. In contrast, XFir uses hashing only at the first level and uses tries for the remaining levels to reduce memory consumption. Moreover, SHIP [58] is designed for Internet IPv6 lookup and exploits the allocation patterns of Internet prefixes by first hashing the leading 23 bits and then dividing the remaining prefix lengths, thereby reducing the trie height. In contrast, XFir targets cloud networks, where tenants can define prefixes with arbitrary lengths within their VPCs, including prefixes shorter than 23 bits. Therefore, XFir reverses the order by first dividing prefixes by length and then applying hashing within each group.

Beyond trie-based lookup, algorithms based on binary search [24, 62, 63] achieve high lookup performance through precomputation, but have worse update complexity than trie-based methods. Moreover, range-based algorithms [27, 56, 57, 60, 70] formulate LPM as querying the smallest range (longest prefix) that contains a given point (IP address), but still incur less efficient prefix updates. Since we target cloud networks where ACLs and SGs can be updated by multiple tenants on a host server, the prefix update efficiency is critical. In addition, a line of work improves LPM by exploiting hardware resources such as TCAMs [21, 55, 72] or GPUs [19, 54]. In Xline, TCAM is scarce and reserved for the tables in CNP pipeline, while GPUs are not available in our case.

7 Conclusion

We propose XFir, the first hardware acceleration mechanism for the host-server new-flow setup in the slow path. XFir is built on our next-generation DPU, Xline, which integrates a fixed-function ASIC, a CPU, and a CNP. In XFir, we redesign the datapath and table layout for new-flow setup in the CNP, optimize the LPM lookup algorithms, and introduce CPU–CNP collaboration mechanisms. XFir achieves high new-flow throughput (>776K CPS) and low latency (11.7μs median) while preserving flexibility for continuous feature evolution. Moreover, XFir is practical for large-scale deployment: it requires only one DPU per host server, resulting in low deployment cost.

Acknowledgments

We sincerely thank the anonymous shepherd and reviewers for their insightful feedback. We also thank the software engineers, network architects, and senior management at Tencent for their support and effort to develop and deploy XFir. Congcong Miao is the corresponding author.

References

[1] AMD. 2026. AMD Alveo Adaptable Accelerator Cards. Retrieved Jun. 2026 from https://www.amd.com/en/products/accelerators/alveo.html

[2] AMD. 2026. AMD FPGAs. Retrieved Jun. 2026 from https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga.html

[3] AMD. 2026. AMD Pensando DPU Technology. Retrieved Jun. 2026 from https://www.amd.com/en/products/data-processing-units/pensando.html

[4] Hirochika Asai and Yasuhiro Ohara. 2015. Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup. In *Proceedings of ACM SIGCOMM'15*. ACM, 57–70.

[5] AWS. 2026. Cloud Object Storage - Amazon S3. Retrieved Jun. 2026 from https://aws.amazon.com/s3/

[6] Masanori Bando, Yi-Li Lin, and H. Jonathan Chao. 2012. FlashTrie: Beyond 100-Gb/s IP Route Lookup Using Hash-Based Prefix-Compressed Trie. *IEEE/ACM Transactions on Networking* 20, 4 (2012), 1262–1275.

[7] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: a Protected Dataplane Operating System for High Throughput and Low Latency. In *Proceedings of USENIX OSDI'14*. USENIX, 49–65.

[8] Ash Bhalgat. 2021. Accelerating Connection Tracking to Turbo-Charge Stateful Security. Retrieved Jun. 2026 from https://developer.nvidia.com/blog/accelerating-connection-tracking-to-turbo-charge-stateful-security/

[9] 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 *Proceedings of ACM SIGCOMM'13*. ACM, 99–110.

[10] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, and Zeke Wang. 2024. Demystifying Datapath Accelerator Enhanced Off-path SmartNIC. In *Proceedings of IEEE ICNP'24*. IEEE, 1–12.

[11] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. 1997. Small Forwarding Tables for Fast Routing Lookups. In *Proceedings of ACM SIGCOMM'97*. ACM, 3–14.

[12] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. 2003. Longest Prefix Matching Using Bloom Filters. In *Proceedings of ACM SIGCOMM'03*. ACM, 201–212.

[13] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. 2006. Longest Prefix Matching Using Bloom Filters. *IEEE/ACM Transactions on Networking* 14, 2 (2006), 397–409.

[14] DPDK. 2026. DPDK - The Open Source Data Plane Development Kit Accelerating Network Performance. Retrieved Jun. 2026 from https://www.dpdk.org/

[15] Will Eatherton, George Varghese, and Zubin Dittia. 2004. Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates. *ACM SIGCOMM Computer Communication Review* 34, 2 (2004), 97–122.

[16] 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 *Proceedings of USENIX NSDI'18*. USENIX, 51–66.

[17] Rohan Gandhi, Hongqiang Harry Liu, Y. Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2014. Duet: Cloud Scale Load Balancing with Hardware and Software. In *Proceedings of ACM SIGCOMM'14*. ACM, 27–38.

[18] Pankaj Gupta, Steven Lin, and Nick McKeown. 1998. Routing Lookups in Hardware at Memory Access Speeds. In *Proceedings of IEEE INFOCOM'98*. IEEE, 1240–1247.

[19] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: a GPU-Accelerated Software Router. In *Proceedings of ACM SIGCOMM'10*. ACM, 195–206.

[20] Yutaro Hayakawa, Michio Honda, Douglas Santry, and Lars Eggert. 2021. Prism: Proxies without the Pain. In *Proceedings of USENIX NSDI'21*. USENIX, 535–549.

[21] Tsunemasa Hayashi and Toshiaki Miyazaki. 1999. High-speed Table Lookup Engine for IPv6 Longest Prefix Match. In *Proceedings of IEEE GLOBECOM'99*. IEEE, 1576–1581.

[22] Intel. 2026. Intel Tofino Series. Retrieved Jun. 2026 from https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[23] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: a Highly Scalable User-Level TCP Stack for Multicore Systems. In *Proceedings of USENIX NSDI'14*. USENIX, 489–502.

[24] Donghong Jiang, Yanbiao Li, Yuxuan Chen, Jing Hu, Yi Huang, and Gaogang Xie. 2025. Heuristic Binary Search: Adaptive and Fast IPv6 Route Lookup With Incremental Prefix Updates. *IEEE Transactions on Networking* 33, 2 (2025), 554–569.

[25] Antoine Kaufmann, Simon Peter, Naveen Kr Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of ACM ASPLOS'16*. ACM, 67–81.

[26] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. TEA: Enabling State-Intensive Network Functions on Programmable Switches. In *Proceedings of ACM SIGCOMM'20*. ACM, 90–106.

[27] Butler Lampson, Venkatachary Srinivasan, and George Varghese. 1999. IP Lookups Using Multiway and Multicolumn Search. *IEEE/ACM Transactions on Networking* 7, 3 (1999), 324–334.

[28] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M Swift, and TV Lakshman. 2017. UNO: Uniflying Host and SmartNIC Offload for Flexible Packet Processing. In *Proceedings of ACM SoCC'17*. ACM, 506–519.

[29] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. Clicknp: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proceedings of ACM SIGCOMM'16*. ACM, 1–14.

[30] Xing Li, Xiaochong Jiang, Ye Yang, Lilong Chen, Yi Wang, Chao Wang, Chao Xu, Yilong Lv, Bowen Yang, Taotao Wu, Haifeng Gao, Zikang Chen, Yisong Qiao, Hongwei Ding, Yijian Dong, Hang Yang, Jianming Song, Jianyuan Lu, Pengyu Zhang, Chengkun Wei, Zihui Zhang, Wenzhi Chen, Qinming He, and Shunmin Zhu. 2024. Triton: A Flexible Hardware Offloading Architecture for Accelerating Apsara vSwitch in Alibaba Cloud. In *Proceedings of ACM SIGCOMM'24*. ACM, 750–763.

[31] Xiaofeng Lin, Yu Chen, Xiaodong Li, Junjie Mao, Jiaquan He, Wei Xu, and Yuanchun Shi. 2016. Scalable Kernel TCP Design and Implementation for Short-lived Connections. In *Proceedings of ACM ASPLOS'16*. ACM, 339–352.

[32] Linux man-pages project. 2020. io_uring(7) - Linux Manual Page. Retrieved Jun. 2026 from https://man7.org/linux/man-pages/man7/io_uring.7.html

[33] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3:Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *Proceedings of USENIX ATC'19*. USENIX, 363–378.

[34] Jianyuan Lu, Shunmin Zhu, Jun Liang, Yuxiang Lin, Tian Pan, Yisong Qiao, Yang Song, Wenqiang Su, Yixin Xie, Yanqiang Li, Enge Song, Shize Zhang, Xiaoqing Sun, Rong Wen, Xionglie Wei, Biao Lyu, and Xing Li. 2025. Albatross: A Containerized Cloud Gateway Platform with FPGA-accelerated Packet-level Load Balancing. In *Proceedings of ACM SIGCOMM'25*. ACM, 71–84.

[35] 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 *Proceedings of ACM SOSP'19*. ACM, 399–413.

[36] Congcong Miao, Zhiyi Yao, Jianchao Lv, Jinglin Wang, Shihan Lin, Xinyi Zhang, Yunming Xiao, Wei Guo, Jiwu Bu, Yachen Wang, Marco Canini, Xianneng Zou, and Gaogang Xie. 2026. Cost-Effective and Reliable Global Internet Peering with Programmable Switches. In *Proceedings of USENIX NSDI'26*. USENIX, 2671–2684.

[37] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of ACM SIGCOMM'17*. ACM, 15–28.

[38] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In *Proceedings of USENIX NSDI'20*. USENIX, 77–92.

[39] Donald R. Morrison. 1968. PATRICIA-Practical Algorithm To Retrieve Information Coded in Alphanumeric. *Journal of ACM (JACM)* 15, 4 (1968), 514–534.

[40] Stefan Nilsson and Gunnar Karlsson. 1999. IP-Address Lookup Using LC-Tries. *IEEE Journal on Selected Areas in Communications* 17, 6 (1999), 1083–1092.

[41] NVIDIA. 2020. What Is a DPU? Retrieved Jun. 2026 from https://blogs.nvidia.com/blog/whats-a-dpu-data-processing-unit/

[42] NVIDIA. 2026. DPA Subsystem. Retrieved Jun. 2026 from https://networking-docs.nvidia.com/doca/sdk/dpa-subsystem

[43] NVIDIA. 2026. NVIDIA BlueField Platform. Retrieved Jun. 2026 from https://www.nvidia.com/en-us/networking/products/data-processing-unit/

[44] NVIDIA. 2026. Virtual Switch on BlueField: Connection Tracking Offload. Retrieved Jun. 2026 from https://networking-docs.nvidia.com/doca/sdk/virtual-switch-on-bluefield#Connection-Tracking-Offload

[45] Tian Pan, Kun Liu, Xionglie Wei, Yisong Qiao, Jun Hu, Zhiguo Li, Jun Liang, Tiesheng Cheng, Wenqiang Su, Jie Lu, Yuke Hong, Zhengzhong Wang, Zhi Xu, Chongjing Dai, Peiqiao Wang, Xuetao Jia, Jianyuan Lu, Enge Song, Jun Zeng, Biao Lyu, Ennan Zhai, Jiao Zhang, Tao Huang, Dennis Cai, and Shunmin Zhu. 2024. LuoShen: A Hyper-Converged Programmable Gateway for Multi-Tenant Multi-Service Edge Clouds. In *Proceedings of USENIX NSDI'24*. USENIX, 877–892.

[46] Tian Pan, Enge Song, Yueshang Zuo, Shaokai Zhang, Yang Song, Jiangu Zhao, Wengang Hou, Jianyuan Lu, Xiaoqing Sun, Shize Zhang, Ye Yang, Jiao Zhang, Tao Huang, Biao Lyu, Xing Li, Rong Wen, Zhigang Zong, and Shunmin Zhu. 2025.

Hermes: Enhancing Layer-7 Cloud Load Balancers with Userspace-Directed I/O Event Notification. In *Proceedings of ACM SIGCOMM'25*. ACM, 363–380.

[47] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, Enge Song, Jiao Zhang, Tao Huang, and Shunmin Zhu. 2021. Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches. In *Proceedings of ACM SIGCOMM'21*. ACM, 194–206.

[48] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-Accelerated Network Applications. In *Proceedings of USENIX OSDI'18*. USENIX, 663–679.

[49] Kun Qian, Sai Ma, Mao Miao, Jianyuan Lu, Tong Zhang, Peilong Wang, Chenghao Sun, and Fengyuan Ren. 2019. FlexGate: High-Performance Heterogeneous Gateway in Data Centers. In *Proceedings of ACM APNet'19*. ACM, 36–42.

[50] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostić, and Marco Chiesa. 2023. A High-Speed Stateful Packet Processing Approach for Tbps Programmable Switches. In *Proceedings of USENIX NSDI'23*. USENIX, 1237–1255.

[51] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proceedings of ACM SIGCOMM'17*. ACM, 418–431.

[52] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism. In *Proceedings of USENIX NSDI'22*. USENIX, 87–102.

[53] Xiaoyi Shi, Lin He, Jiasheng Zhou, Yifan Yang, and Ying Liu. 2025. Miresga: Accelerating Layer-7 Load Balancing with Programmable Switches. In *Proceedings of ACM WWW'25*. ACM, 2424–2434.

[54] Veeramani Sonai, Indira Bharathi, Samah Alshathri, Walid El-Shafai, and Mohd Fadzil Abdul Kadir. 2026. High Performance IP Lookup through GPU Acceleration to Support Scalable and Efficient Routing in Data Driven Communication Networks. *Scientific Reports* (2026).

[55] Tian Song, Tianlong Li, and Yating Yang. 2025. PtCAM: Scalable High-Speed Name Prefix Lookup using TCAM. In *Proceedings of ACM SIGCOMM'25*. ACM, 707–719.

[56] Ioannis Sourdis and Sri Harsha Katamaneni. 2011. Longest Prefix Match and Updates in Range Tries. In *Proceedings of IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'11)*. IEEE, 51–58.

[57] Ioannis Sourdis, Georgios Stefanakis, Ruben de Smet, and Georgi N. Gaydadjiev. 2009. Range Tries for Scalable Address Lookup. In *Proceedings of ACM/IEEE Architectures for Networking and Communications Systems (ANCS'09)*. ACM/IEEE, 143–152.

[58] Thibaut Stimpfling, Normand Belanger, J. M. Pierre Langlois, and Yvon Savaria. 2019. SHIP: A Scalable High-Performance IPv6 Lookup Algorithm That Exploits Prefix Characteristics. *IEEE/ACM Transactions on Networking* 27, 4 (2019), 1529–1542.

[59] Hai Sun, Yan Sun, Victor C. Valgenti, and Min Sik Kim. 2014. A Hierarchical Hashing Scheme to Accelerate Longest Prefix Matching. In *Proceedings of IEEE GLOBECOM'14*. IEEE, 1296–1302.

[60] Subhash Suri, George Varghese, and Priyank Ramesh Warkhede. 2001. Multiway Range Trees: Scalable IP Lookup with Fast Updates. In *Proceedings of IEEE GLOBECOM'01*. IEEE, 1610–1614.

[61] Zahid Ullah, Manish K Jaiswal, Ray CC Cheung, and Hayden KH So. 2015. UE-TCAM: an ultra efficient SRAM-based TCAM. In *2015 IEEE Region 10 Conference (TENCON'15)*. IEEE, 1–6.

[62] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. 1997. Scalable High Speed IP Routing Lookups. In *Proceedings of ACM SIGCOMM'97*. ACM, 25–36.

[63] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. 2001. Scalable High-Speed Prefix Matching. *ACM Transactions on Computer Systems (TOCS)* 19, 4 (2001), 440–482.

[64] Yunming Xiao, Yinchao Yang, Jiaqi Zheng, Xuqian Li, Dongbo Gu, Jun Zhang, Miantao Wan, Chao Pei, Chen Tian, Mingwei Xu, Ang Chen, and Congcong Miao. 2026. CubeTrace: Microscopic Network Tracing for Heterogeneous Cloud Gateways. In *Proceedings of ACM SIGCOMM'26*. ACM.

[65] Tong Yang, Gaogang Xie, Yanbiao Li, Qiaobin Fu, Alex X. Liu, Qi Li, and Laurent Mathy. 2014. Guarantee IP Lookup Performance with FIB Explosion. In *Proceedings of ACM SIGCOMM'14*. ACM, 39–50.

[66] Tong Yang, Gaogang Xie, Alex X. Liu, Qiaobin Fu, Yanbiao Li, Xiaoming Li, and Laurent Mathy. 2018. Constant IP Lookup With FIB Explosion. *IEEE/ACM Transactions on Networking* 26, 4 (2018), 1821–1836.

[67] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proceedings of ACM SIGCOMM'17*. ACM, 432–445.

[68] Heng Yu, Jian Wang, Jian Zhao, Kai Ren, Guozhi Lin, Baozeng Zhang, Yunpeng Guan, Xin Li, Hao Yin, Jiajun Liang, Liang Wang, Chao Pei, Yachen Wang, Xin Jin, Jilong Wang, and Congcong Miao. 2025. Fornax: A Hardware-Centric Session Management in Large Public Cloud Network. In *Proceedings of ACM SIGCOMM'25*. ACM, 663–676.

[69] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, Xiongfei Geng, Tao Feng, Feng Ning, Kai Chen, and Chuanxiong Guo. 2022. Tiara: A Scalable and Efficient Hardware Acceleration Architecture for Stateful Layer-4 Load Balancing. In *Proceedings of USENIX NSDI'22*. USENIX, 1345–1358.

[70] Zhihao Zhang, Lanzheng Liu, Chen Chen, Huiba Li, Jiwu Shu, Windsor Hsu, and Yiming Zhang. 2026. PlanB: Efficient Software IPv6 Lookup with Linearized B+-Tree. In *Proceedings of USENIX NSDI'26*. USENIX, 1005–1019.

[71] Chenxingyu Zhao, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2025. White-Boxing RDMA with Packet-Granular Software Control. In *Proceedings of USENIX NSDI'25*. USENIX, 427–449.

[72] Kai Zheng, Chengchen Hu, Hongbin Lu, and Bin Liu. 2004. An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine. In *Proceedings of IEEE INFOCOM'04*. IEEE, 1984–1994.