FlexSFP: Rethinking Network Intelligence Inside the Cable

Angelo Tulumello
CNIT/Axbryd
Rome, Italy

Alessandro Rivitti
University of Rome Tor Vergata/Axbryd
Rome, Italy

Giacomo Belocchi
University of Rome Tor Vergata/Axbryd
Rome, Italy

Giuseppe Bianchi
University of Rome Tor Vergata
Rome, Italy

Luka Perkov
Bit Ltd
Zagreb, Croatia

Abstract

As network programmability promotes increasingly intelligent hosts, NICs, and switches, we argue that the next frontier lies in making the cables themselves smarter. By embedding lightweight, wire-speed processing directly within transceivers, we enable precise, low-latency, and energy-efficient network functions, providing a modular and cost-effective path to extend the life of legacy infrastructure and to fundamentally rethink where intelligence belongs in the data plane. To concretize such vision, this paper introduces FlexSFP, a programmable variant of the standard SFP+ transceiver, embedding lightweight FPGA-based logic for in-line packet processing. Driven by both technical and economic considerations, FlexSFP offers a compelling "cheap path" alongside host CPUs and SmartNICs, enabling targeted acceleration where cost, power, and deployment flexibility are critical. We explore practical use cases, such as per-port firewalling, in-line telemetry, and retrofitting legacy switches, and outline a feasible architecture and programming model.

Keywords

Programmable Networks, SFP Modules, FPGA, Packet Processing, Network Hardware, Smart Transceivers, Data Plane Offloading

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/2025/11
https://doi.org/10.1145/3772356.3772427

1 Introduction

Modern networks face increasing demands for performance, flexibility, and efficiency. While control-plane programmability has matured through software-defined networking (SDN), and data-plane acceleration has advanced with the adoption of SmartNICs and DPUs, these solutions remain costly and power-intensive. Moreover, they are frequently deployed to handle simple, repetitive tasks (e.g., filtering, tagging, etc.) that do not require such powerful hardware, leading to over-provisioning that is economically and architecturally difficult to justify.

Yet acceleration and offloading remain essential to achieving high-performance network operation. We argue that these capabilities should no longer be limited to hosts or specialized NICs, but should extend into the termination points of the network cables themselves. At this physical boundary lies a natural and underutilized candidate: the *Small Form-factor Pluggable module* (SFP). Ubiquitous across network infrastructure, SFPs terminate optical links, populate every port of every switch and router, and serve as the primary ingress and egress points of the data plane. As such, moving computation directly into the transceiver itself appears to us as a natural next step in the evolution of network programmability.

We introduce FlexSFP, a programmable SFP module embedding lightweight logic (e.g., FPGA) within standard pluggable optics. This enables high-frequency operations—filtering, tagging, and telemetry—to be performed at wire speed, with minimal energy cost, directly at the physical network edge. In essence, beyond the traditional slow path (CPU) and fast path (SmartNIC), programmable SFP modules represent a natural place to realize a third, *complementary*, **cheap path**: a low-power, low-cost tier for in-line processing of ubiquitous micro-tasks, deployable pervasively and incrementally across the network.

Crucially, FlexSFP can refactor legacy switches into intelligent, policy-enforcing edge devices—without replacing switch hardware or modifying control software. One can

envision FlexSFP as the SmartNIC of the switch: a compact, programmable datapath element embedded at every port, capable of executing common packet-processing functions inline. Just as SmartNICs brought intelligence to the server edge, FlexSFP brings modular, cost-efficient computation to *legacy* switching fabrics, unlocking new functionality without costly infrastructure upgrades.

In the remainder of this paper, we outline our motivations and vision in Section 2, followed by use cases in Section 3. Section 4 describes the proposed architecture and programming model, while Section 5 presents a preliminary feasibility evaluation based on a simple NAT implementation. We conclude with open research challenges in Section 6.

2 Motivation

Modern SmartNICs [2–4, 29, 42] are powerful, general-purpose accelerators featuring CPU clusters, substantial memory, and programmable pipelines. They are designed for demanding workloads such as full network virtualization [13, 22], advanced storage protocols [18, 28], and data aggregation for AI [40, 49]. While well justified in hyperscale and high-performance computing environments, these capabilities may be disproportionate for simpler tasks such as packet filtering, VLAN tagging, or basic telemetry, where their use can entail unnecessary cost, high power consumption, and suboptimal resource utilization. This situation exposes a significant *acceleration gap*, as network operators are often left to choose between two suboptimal options:

- executing simple tasks on the host CPU, reintroducing latency, jitter, and resource contention—the very issues offloading was intended to avoid, or
- deploying a full-featured SmartNIC, incurring additional cost and power consumption for capabilities that may remain largely unused.

FlexSFPs are conceived to fill this gap by offering a focused, lightweight processing platform for the "long tail" of simple, recurring packet-processing tasks, executed directly at the optical edge, without the complexity or cost of a SmartNIC or DPU. Beyond addressing architectural inefficiencies, this approach also responds to two critical pressures in modern network design: cost and power (see also Section 5 for further quantitative cost and energy trade-offs).

Economic Drivers: The cost gap between conventional and accelerated network interfaces is substantial. A modern SmartNIC or DPU can add $800 to over $2,000 per port to a server's total cost, making widespread deployment economically impractical. In contrast, one FlexSFP is expected to add only a modest premium over standard SFP+ or QSFP modules, which typically range from as little as less than $10 to several hundred dollars. This enables a more scalable and cost-effective deployment model, bringing programmability to a much broader portion of the network, not just to its most critical or centralized nodes.

Energy Efficiency: Power consumption poses a similar constraint. A single SmartNIC may draw between 25 and 75 watts per port, contributing significantly to a rack's thermal and power budget. By comparison, a FlexSFP is designed to stay within the 1–3W envelope of a standard transceiver, achieving an order-of-magnitude reduction in power usage. This low power profile enables deployment in edge environments with tight thermal limits, where traditional acceleration platforms are infeasible.

2.1 Refactoring legacy switches

A strategically significant benefit of the FlexSFP is its ability to decouple feature deployment from hardware replacement cycles. Adding support for new tunneling formats, telemetry, or access policies typically requires replacing entire network devices, indeed an expensive and disruptive process. Rather, programmable SFPs enable a modular, plug-and-play approach, allowing operators to extend the capabilities of legacy infrastructure without modifying switch hardware or control software.

A compelling application of the FlexSFP architecture is in large-scale telecom deployments, where operators manage thousands of legacy aggregation switches across urban and suburban networks. These fixed-function L2/L3 devices connect residential fiber (FTTH) or mobile base stations to metro cores but lack programmability, telemetry, and in-line enforcement capabilities. As a result, per-subscriber policies such as IPv6 filtering, DoH blocking, or basic rate-limiting must be enforced upstream, introducing latency and added complexity. The lack of local observability further prevents early detection of congestion or spoofing, and makes troubleshooting difficult.

Upgrading these switches with external SmartNICs is economically and operationally impractical: it would require full hardware replacement, at significant capital and energy cost, and is incompatible with fanless, space-constrained enclosures common at the edge. In contrast, replacing the existing SFP modules with programmable SFPs offers a modular, drop-in upgrade path. Each port becomes a programmable enforcement point, capable of inline filtering, VLAN tagging, flow marking, or telemetry export, without any modification to the chassis or switch OS. This decoupled, drop-in upgrade model allows legacy switches to be refactored into intelligent edge nodes, and enables operators to selectively extend the functionality of existing infrastructure, deploy new services incrementally, and avoid the cost, disruption, and rigidity of monolithic hardware refresh cycles.

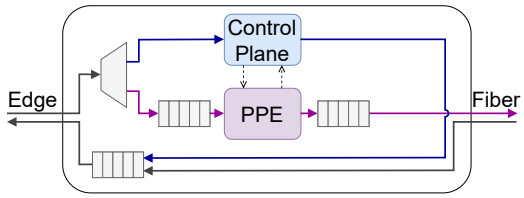
3 Use Cases and Scenarios

To illustrate the practical potential of programmable SFPs, in this section we outline a (non-exhaustive) spectrum of use cases, which aim to show how lightweight, wire-speed programmability in SFP modules may enable deployments that are impractical or uneconomical with SmartNICs or centralized appliances.

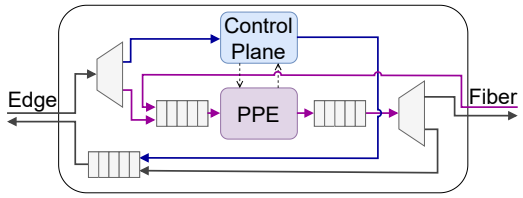
Security and Policy Enforcement. Filtering is a classic use case: access control lists (ACLs) are needed at almost every edge point, yet are commonly offloaded to SmartNICs, fixed-function appliances, or implemented inefficiently on host CPUs. With programmable SFPs, packet filtering and firewalling can occur directly at the optical edge, dropping traffic before it reaches the NIC, the switch, or even the customer premises. This architectural placement naturally supports a clean separation between security and service logic, allowing traffic to be screened before reaching service endpoints. Inline security use cases may also include packet sanitization and protocol validation, such as removing deprecated headers, blocking malformed packets, or rate-limiting traffic from selected sources. The inline feasibility of these primitives in the data plane has been demonstrated in programmable switches using P4, see e.g., P4DDPI's DNS-based filtering [1] and Nimble's in-network rate limiting [19].

Packet Transformation and Forwarding. Many data-plane operations, such as header encapsulation, VLAN tagging, or flow-based load balancing, are repetitive, predictable, and amenable to offload. These transformations are a perfect target for programmable SFPs: they are simple, do not require deep buffers or complex scheduling, and can be executed at line rate directly at the network edge. For example, programmable SFPs can insert tunneling headers for GRE, VXLAN, or IP-in-IP without involving the host, or apply VLAN tagging and QinQ for L2 segmentation in legacy environments. Load balancing is another natural fit, such as hashing over packet headers to distribute flows across uplinks, similar to Katran [15], but executed directly at the optical boundary. Unlike traditional deployments where a SmartNIC or host CPU performs rerouting in software, implementing this logic within the SFP simplifies the datapath and removes need for such external components. Besides, programmable hardware platforms like FlowBlaze [32] and Domino [44] have shown that even more advanced stateful forwarding logic can be achieved at line rate using compact, match-action logic.

Monitoring and Observability. Embedding telemetry logic in the SFP allows in-line timestamping, labeling, or sampling even on legacy switches without native support. For instance, a FlexSFP could export NetFlow-like stats or insert lightweight metadata for in-band measurements, similar to what has been demonstrated in in-band network telemetry (INT) [8, 46]. These techniques bring modern observability to infrastructure that can't otherwise be instrumented, without incurring high overhead. In addition to passive monitoring, programmable SFPs can also play an active role in detecting faults such as link flapping, microbursts, or fiber breaks, with a "wire-level" capillarity that centralized tools can hardly achieve.



(a) One-Way-Filter architecture



(b) Two-Way-Core architecture

Figure 1: Alternative architectures for FlexSFP packet processing logic.

Edge Acceleration. At the edge of mobile networks, links between Radio Units (RUs), Distributed Units (DUs), and Centralized Units (CUs) often carry baseband data over Ethernet, encapsulating digitized PHY signals. Especially in micro-cell deployments, these links form a dense, time-sensitive, distributed infrastructure with minimal space for compute. Recent work like Janus [17] shows that telemetry and control tasks (throughput estimation, handover analysis, cell occupancy tracking) can be performed with low overhead, suggesting the feasibility of deploying similar *codelets* at the optical edge. Similarly, in Passive Optical Networks (PONs), programmable optical terminals could shape, classify, or drop traffic directly at the fiber edge, and enforce per-user SLAs, tag VoIP streams, or apply early traffic policing in multi-tenant access networks without upgrading Optical Line Terminals (OLT) hardware or customer routers.

4 FlexSFP design

We begin by examining the implications of embedding programmability within the constrained physical footprint of an SFP module. Enabling such devices to execute meaningful network functions requires a clear understanding of their architectural and operational limitations. Specifically, we identify key constraints common to embedded hardware systems: *Area*, *Power*, and *Thermal dissipation*. To support

programmability, we need to accommodate, alongside the SFP components, a hardware chip such as a processor, an FPGA, or a mixed system like a System on Chip (SoC).

This integration must be achieved within the tight size and layout constraints of the SFP form factor, making efficient Printed Circuit Board (PCB) utilization a key design challenge. While a detailed analysis of power and thermal limits across SFP generations (e.g., SFP+, QSFP, OSFP) is left for future work, it is worth noting that commercial SFP and SFP+ modules already exist with embedded low-performance processors [31], or FPGAs [45], though, to the best of our knowledge, only used for fixed-function tasks. For the FlexSFP prototype, we used an off-the-shelf commercial product based on the Microchip reference design [26].

4.1 Architecture

We now outline a component-level architecture for a programmable SFP, guided by functional requirements and implementation constraints. We identify three possible logical architectures that differentiate mainly by complexity and type of packet processing needed in different scenarios. It is worth noting that an SFP device has in itself two separate networking interfaces, one connected to the host system (identified as the Edge Connector in Figure 1) and the other one typically connected to the fiber optical connectors. Placing programmability in between these two interfaces means that an application could either be meant for unidirectional traffic in either direction or bidirectional traffic.

To support runtime programmability rather than fixed-function deployment, a lightweight control plane is essential, enabling runtime control plane operations and over-the-air reprogramming of packet processing logic.

With this in mind, we introduce a preliminary architecture, referred to as One-Way-Filter, illustrated in Figure 1a. In this design, the packet processing logic is placed only on the path from the edge interface toward the optical link via a *Packet Processing Engine* (PPE as referred to in the figures) which is better detailed in §4.2. Traffic arriving from the edge is demultiplexed: control-plane packets are directed to the management core, while all other packets are forwarded through the PPE. In the reverse direction, packets arriving from the optical link are merged with control-plane traffic and forwarded to the edge. We assume that control-plane traffic is negligible compared to the data-plane traffic traversing the module, such that the aggregation step does not become a performance bottleneck and does not impact overall throughput.

If applications are expected to interact with packets from both interfaces, a modified version of the *One-Way-Filter* architecture can be employed. We refer to this enhanced design as the *Two-Way-Core*, illustrated in Figure 1b. In this architecture, packets arriving from both the edge and optical interfaces are first aggregated and then passed through the PPE. After processing, packets are demultiplexed and forwarded to the appropriate egress interface based on application logic. This design introduces two key considerations:

- **Processing Load:** Aggregating traffic from both interfaces effectively doubles the packet rate entering the processing engine. To maintain line-rate throughput on both ports, one feasible approach is to increase the operating frequency of the packet processing engine while keeping the rest of the system components clocked at their original rates.
- **Hardware Overhead:** Although the *Two-Way-Core* architecture incurs a higher hardware footprint than the *One-Way-Filter*, the increase is not linear. Shared components mitigate the growth in required resources.

It is worth noting that in the *One-Way-Filter*, the PPE could be placed in either direction (optical to electrical or vice-versa) or even both if hardware resources suffice.

A third architectural model envisions integrating a dedicated control-plane network interface into the SFP, effectively promoting the control plane from a passive management entity to an active participant in the data path. In this design, the control plane is not limited to configuring the data plane, but can also originate and terminate traffic, transforming the SFP from a reactive device into an active network component capable of generating traffic.

This shift opens up new possibilities: if lightweight application logic could be embedded directly into the control plane, the SFP could act as a self-contained microservice node. Such a model would require a more capable control plane architecture, moving beyond minimal footprint designs toward more feature-rich platforms.

Control Plane Considerations. To support this, a programmable SFP must include an embedded control plane. Even in lightweight designs, a minimal control stack is needed for configuration, monitoring, and runtime coordination. We identify two main classes of embedded control logic:

- **SoC-based designs**, which embed full-featured ARM (or RISC-V) hard processors alongside the dataplane logic. These allow running standard OSes like Linux, enabling support for complex services such as RPC protocols or REST APIs. While more expensive and power-hungry, SoC-based programmable SFPs are well-suited for environments that require flexible reconfiguration or interaction with external orchestration systems.
- **Softcore-based designs**, which embed minimal RISC-V or MIPS CPUs as logic blocks within the FPGA fabric, programmed with lightweight OSes like FreeRTOS [14] or Zephyr [47]. These CPUs are resource-constrained but are sufficient for many of the use cases described in

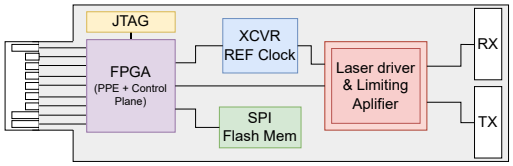


Figure 2: Prototype for a programmable SFP with Microchip FPGA (MPF200T).

§ 3. In particular, use cases such as firewalling, tunneling, or in-line telemetry often require only static rule loading or coarse-grained updates that can be coordinated via simple control messages.

In both cases, the FlexSFP must expose a basic network-accessible control interface. As shown in Figure 1, the on-board arbiter connects the programmable dataplane with the host interface and the out-of-band management port, allowing remote access to the control logic without disrupting the dataplane. This is essential for centralized orchestration across a fleet of FlexSFPs, while preserving the independence of per-port behavior.

4.2 Programming model

In order to provide programmability at speed, we believe that an FPGA chip, or at least some FPGA fabric inside a SoC, is needed. Mainly because in such an embedded environment, one cannot simply scale up performance linearly with the number of general-purpose cores due to limitations in both thermal constraints and chip area footprint.

We envision programmable SFPs being used by selecting one of the proposed architecture shells and embedding custom applications within the PPE. Application development may follow various approaches, including (but not limited to): using High-Level Synthesis (HLS) tools for C-like programs [5, 23] or ML models [9] to generate IP cores; designing custom pipelines based on RMT [10, 12, 21, 43, 44]; or implementing offload mechanisms for eBPF/XDP [6, 11, 38].

In the FlexSPF workflow, the developer writes the packet function (e.g., an XDP program). An HLS toolchain converts it to HDL and generates an IP core. The build framework integrates this into an architecture shell, finalizes clocks, memory, and IO, and emits the SFP bitstream. During prototype phase, the bitstream is loaded via JTAG, while in production artifacts are deployed remotely.

Programmability is layered: the bitstream fixes the datapath, while a minimal embedded control plane exposes APIs to read/write tables and counters with atomic, runtime updates at line rate. We also envision re-programmability over the network: the control plane authenticates reconfiguration packets whose payload carries a new bitstream; a small FSM writes it to SPI flash and then triggers a reboot so the SFP boots the new application.

	4LUT	FF	uSRAM	LSRAM
Mi-V	8696	376	6	4
Elec. I/F	6824	6924	118	0
Opt. I/F	6813	6924	118	0
NAT app	9122	11294	36	160
Used	**31455**	**25518**	**278**	**164**
Avail.	**192408**	**192408**	**1764**	**616**
Perc.	**16%**	**13%**	**15%** (≈20kb)	**26%** (≈4Mb)

Table 1: Resource usage for the simple NAT case study, broken down by design component

4.3 Prototype

A prototype SFP+ [26] module has been developed featuring a mid-range PolarFire® FPGA (MPF200T-FCSG325 [27]). The FPGA offers approximately 200k logic elements and includes 13.3Mb of on-chip SRAM. As shown in Figure 2, the module integrates a 128Mb SPI flash, two bidirectional 12.7Gbps high-speed transceivers (one corresponding to the electrical interface and the other to the optical one), and a JTAG bus. The flash memory is such that multiple designs could be stored, enabling the module to be reconfigurable at runtime while the JTAG bus is mainly meant for initial prototyping.

5 Preliminary Evaluation

This section provides a concrete feasibility snapshot by describing the resources required to implement a simple NAT application inside a programmable SFP. Our goal is to offer an initial, grounded perspective on what is achievable within the constraints of such a compact, power-limited platform.

We begin by outlining the implementation of a minimal NAT function in the FlexSFP prototype (§5.1), followed by a discussion of its resource footprint, including power consumption, performance, and hardware cost (§5.2).

5.1 A simple NAT case study

To demonstrate the practical viability of the FlexSFP concept, we coded a simple static Network Address Translation (NAT) application and tested it in a prototype board like the one described in §4.3. The goal was to implement a basic one-to-one NAT function, capable of translating source IP addresses for outgoing traffic at 10 Gbps line-rate. Due to the simplicity of the networking application it is often handled by software or larger appliances, making it an ideal candidate for lightweight hardware offloading.

Implementation. Extending the reference design provided by Microchip [25], in Table 1 is reported the overall resource allocation for our case study implementation. In detail, with *Mi-V* [24] we refer to a simple RISC-V core that we used to implement the lightweight control plane, as discussed in §4, tasked with startup configurations of the transceivers, laser driver and limiting amplifier and the NAT table. Electrical

Use case	Logic	BRAM
FlowBlaze (1 stage)	71 712 LUT6 (≈ 115 k LE)†	14 148
Pigasus [48]	207 960 ALM (≈ 416 k LE)‡	64 400
hXDP (1 core)	≈ 68 689 LUT6 (≈ 109 k LE)†	1 799
ClickNP IPSec GW [20]	≈ 242 592 LUT6 (≈ 388 k LE)†	39 161
FlexSFP (MPF200T)	192 k LE	13 300

† 1 LUT6 ≈ 1.6 LE [7] ‡ 1 ALM ≈ 2 LE [16]

Table 2: FPGA resource usage of key designs; logic normalized to 4-input LE equivalents, BRAM in kbit

and Optical Interfaces encompass respectively for the two 10G Ethernet IpCores needed to translate serial electrical and optical signals into Ethernet packets to pass to the dataplane. The NAT application serves as the Packet Processing Engine for our implementation.

It is worth noting that the NAT uses a basic source IP hash table to store 32,768 flows, which accounts for the high LSRAM usage, while still showing promising potential for larger tables. LSRAM and uSRAM are two types of on-chip memory, respectively holding 20Kb and 64×12b each, hence the need for LSRAM in the NAT. The design has been clocked at 156.25 MHz with a 64b datapath, sufficient for line-rate.

We performed a simple end-to-end test, which confirmed line-rate performance, as the NAT function is stateless.

Comparison. As a comparison of potential resource fit within the FlexSFP, we report in Table 2 four FPGA implementations of network functions found in literature to check whether they could potentially or not fit inside the FlexSFP itself. While these numbers do not account for an extremely accurate analysis due to aspects related to different vendors FPGA devices-fluctuations on the numbers that could depend on synthesis strategies and chosen routing algorithm-we argue that an overall estimate in terms of order of magnitude of Logic Elements shows potential for programmable SFPs to be viable devices for next generation networking.

Power consumption. To assess power consumption and compare a typical SFP with the FlexSFP we used a custom in-house testbed capable of measuring current drawn from a Thunderbolt-connected NIC [33] with a single 10Gbps Ethernet port. Initially, we established a baseline by measuring the power consumption of the NIC without any SFP inserted, which yielded a value of 3.800W. Then we measured a connected SFP while in a stress test of receiving and transmitting linerate traffic, resulting in a measured power draw of 4.693W. Finally, when using the FlexSFP, power consumption increased to 5.320W. Rather than in absolute values, considerations can be drawn from the comparison. While a single SFP draws ∼.9W of power, the FlexSFP shows an increase in ∼.7W accounting for an overall ∼1.5W.

Solution	Raw $	Raw W	$/10G	W/10G
DPU (BF-2)	1.5–2k	75	300–400	15
Many-core (Ag./DSC)	0.8–1.2k	25	100–150	5
FPGA (U25/U50)	>2k	45–75	200–400	7–10
FlexSFP	**250-300**	**1.5**	**250–300**	**1.5**

Table 3: Raw and ideal-scaled cost/power (per 10 Gb/s).

5.2 Cost

Talking about costs can be both confusing and misleading when systems are not compared under consistent assumptions. Drawing inspiration from [39], we are briefly going to assess the costs of a programmable SFP in comparison to SmartNIC-based solutions. Providing an exact price is inherently speculative, as no vendor currently offers such modules at production scale, and key factors such as volume discounts, yield, compliance costs, and vendor margins remain unknown. What we can offer, however, is a breakdown of the cost structure derived from our FlexSFP prototype implementation, suitable for an order-of-magnitude estimate.

The most significant cost driver is the FPGA. According to Microchip Direct, the 192k logic element MPF200T-FCSG325E, which is suitable for our design, is priced at approximately $200 per unit for orders of 1,000 pieces or more [27]. A standards-compliant 10GBASE-SR SFP transceiver is inexpensive at scale (e.g., tier-one OEMs such as QSFPTEK list single-unit prices around $10 and decreasing further in bulk quantities [36]). The remaining components—including the laser driver, voltage regulators, reference oscillator, SPI flash, six-layer PCB, and associated manufacturing steps such as reflow, inspection, and functional testing—lack published volume pricing but are conservatively estimated to add $50–$100 per unit. Summing these contributions yields a direct production cost around $300 per unit, with potential reductions toward $250 as volume increases.

Applying the *ideal-scaling* rule in [39], Table 3 normalizes both capital expense and peak board power to a 10 Gb/s slice. The comparison shows that a DPU such as NVIDIA's BlueField-2, still costs several hundred dollars and dissipates tens of watts [29, 41], while a many-core or FPGA SmartNIC (e.g., Pensando DSC-25, Alveo U25 and U50) remains above $100 and 5 W per 10 Gb/s [2, 3, 42]; the proposed FlexSFP stays in the $250–$300 band and around 1.5 W, confirming a roughly two-thirds CAPEX saving an order-of-magnitude power reduction for lightweight edge workloads.

5.3 Discussion

Scalability A key question is how the FlexSFP architecture scales from the 10Gbps rate of our current SFP+ prototype to higher line rates such as 100Gbps and beyond, where scaling by 10× directly challenges the Packet Processing Engine (PPE) to sustain proportionally higher throughput.This is

typically achieved by adjusting the width of the internal datapath (e.g., from 64-bit to 512-bit or wider) and/or raising the clock frequency if and when possible. Both adjustments require a more powerful FPGA, which in turn leads to three main constraints: physical size, power consumption, and thermal dissipation. On the other hand, however, performance is expected to improve with hardware advancements. The current FlexSFP prototype is built on a mature 28nm FPGA; future iterations will leverage ongoing semiconductor trends to deliver greater processing power and support faster transceivers in a more compact chip. It is worth noting that the evolution of pluggable optical modules is governed by the so called, Multi-Source Agreements (MSAs)[30, 34, 35]. Higher-speed interconnects rely on larger form factors like QSFP, and OSFP. These modules are not only physically larger than a FlexSFP but are also designed with higher power and thermal envelopes.

Failure Recovery. The key consideration is identifying failure points within the FlexSFP architecture. Research demonstrates that VCSELs (Vertical-Cavity Surface-Emitting Lasers) exhibit accelerated wear-out compared to electronic components, with time-to-failure following a lognormal distribution and gradual optical power degradation as the primary failure [37]. This reduced VCSEL lifetime provides strong motivation for designing FlexSFP systems with replaceable optical components. While standard SFPs are replaced entirely when lasers fail (since component costs rival full module prices), higher-cost FlexSFP units justify component-level replacing of individual failed lasers, which becomes economically viable given both the higher price and the known reliability limitations of VCSEL. More favorably, the internal visibility provided by the FlexSFP architecture can expose more detailed insights into the specific fault, such as distinguishing between laser degradation and driver circuit malfunction, facilitating targeted repairs.

SmartNIC vs FlexSFP. The SFP+ form factor and power budget push the design toward FPGAs that physically fit the module and stay within its thermal envelope, while still offering integrated SERDES and enough on-chip resources for compact pipelines with a lightweight control path. Rather than replicating board-level architectures with deep state and complex control planes, FlexSFP targets composed L2–L4 functions—multi-field parse/edit, label/tunnel manipulation, per-packet hashing for steering, and in-band timestamping/telemetry—executed at the optical boundary.

In our MPF200T-based prototype, sustaining bidirectional line rate in the Two-Way-Core typically means processing at double the data rate for maximum usage of the bidirectional traffic keeping chains compact (about 3–4 stages) or modestly increasing the PPE clock. A few RISC-V like cores and a small local DRAM are feasible, but deeply stateful pipelines or very large tables (e.g., DPI/IDS with deep L7 parsing, high-throughput IPsec with large SA sets without dedicated crypto) are out of scope by design and are better placed on SmartNICs or hosts, consistent with their higher CPU/memory budgets.

6 Conclusion and outlook

This paper highlights a promising direction in data-plane programmability: bringing in-line processing directly to the optical edge via programmable transceivers. FlexSFP, realizes this vision by embedding lightweight programmable logic into standard SFP modules. Programmable SFPs introduce a *cheap path* for acceleration, offering a low-cost, low-power, modular alternative for executing simple, high-frequency network functions near the ingress point. We argue this architecture addresses key limitations of traditional approaches, especially in environments where the cost, power, or complexity of full-featured SmartNICs or programmable switches is impractical.

Our goal is not to present a fully matured platform, but rather to spark a broader conversation around this design space. The FlexSFP vision opens up a set of compelling research and engineering questions that we lightly touched in §5.3 however merit further investigation, among which:

- **Performance vs. simplicity:** Are programmable SFPs sufficient for common tasks, and how do they compare to SmartNICs in latency, throughput, and flexibility?
- **Latency overhead:** Which practical impact of introducing processing within the SFP, and when is the trade-off between added latency and early enforcement justified?
- **Architecture choices:** Which platforms (FPGAs, embedded processors, or ASIC pipelines) strike the best balance between power, cost, and programmability?
- **Scalability:** Can this approach be extended to higher-speed and higher-density form factors like QSFP-DD or OSFP while meeting power and thermal constraints?

We hope our work encourages further exploration of how fine-grained, edge-centric programmability can be integrated into legacy infrastructure, without the cost and disruption of full-scale hardware refreshes. FlexSFP represents a small yet strategically placed component with the potential to shift how and where intelligence resides in the network.

Acknowledgments

This work was partially supported by the projects SERICS (SEcurity and RIghts In the CyberSpace - PE00000014) and RESTART (Telecommunications of the Future - PE00000001) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, and by the SNS JU under the EU Horizon Europe programme, grant No. 101192462 (FLECON-6G).

References

[1] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb. P4ddpi: Securing p4-programmable data plane networks via dns deep packet inspection. In *NDSS Symposium 2022*, 2022.

[2] AMD. AMD Alveo U25N SmartNIC. https://www.amd.com/en/products/accelerators/alveo/u25n/a-u25n-p06g-pq-g.html, 2025. Accessed: 2025-07-08.

[3] AMD. AMD Alveo U50 SmartNIC. https://www.amd.com/en/products/accelerators/alveo/u50/a-u50-p00g-pq-g.html, 2025. Accessed: 2025-07-08.

[4] AMD. AMD Alveo™ V80 Compute Accelerator. https://www.amd.com/en/products/accelerators/alveo/v80.html, 2025. Accessed: 2025-07-08.

[5] AMD. AMD Vitis™ HLS. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html, 2025. Accessed: 2025-07-08.

[6] AMD. The Nanotube Compiler and Framework. https://github.com/Xilinx/nanotube, 2025. Accessed: 2025-07-08.

[7] AMD Inc. *7 Series FPGAs Configurable Logic Block User Guide*. AMD Xilinx, March 2023.

[8] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher. Pint: Probabilistic in-band network telemetry. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 662–680, New York, NY, USA, 2020. ACM.

[9] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):1–23, 2018.

[10] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. *SIGCOMM Comput. Commun. Rev.*, 43(4):99–110, Aug. 2013.

[11] M. S. Brunella, G. Belocchi, M. Bonola, S. Pontarelli, G. Siracusano, G. Bianchi, A. Cammarano, A. Palumbo, L. Petrucci, and R. Bifulco. hxdp: Efficient software packet processing on fpga nics. *Commun. ACM*, 65(8):92–100, July 2022.

[12] Y. Feng, Z. Chen, H. Song, Y. Zhang, H. Zhou, R. Sun, W. Dong, P. Lu, S. Liu, C. Zhang, Y. Xu, and B. Liu. Empower programmable pipeline for advanced stateful packet processing. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 491–508, Santa Clara, CA, Apr. 2024. USENIX Association.

[13] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, Apr. 2018. USENIX Association.

[14] FreeRTOS. Real-time operating system for microcontrollers and small microprocessors. https://www.freertos.org/. Accessed: 2025-07-08.

[15] C. Hopps. Katran: A high performance layer 4 load balancer. https://github.com/facebookincubator/katran. Accessed: 2025-07-08.

[16] Intel Corporation. *Adaptive Logic Module (ALM)*, 2024.

[17] A. Kalia, N. Lazarev, L. Xue, X. Foukas, B. Radunovic, and F. Y. Yan. Taking 5G Analytics and Control to a New Level. In *USENIX NSDI*, 2025.

[18] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel. Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 756–771, New York, NY, USA, 2021. Association for Computing Machinery.

[19] W. Kwon, G.-I. Yu, E. Jeong, and B.-G. Chun. Nimble: Lightweight and parallel gpu task scheduling for deep learning. *Advances in Neural Information Processing Systems*, 33:8343–8354, 2020.

[20] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 1–14, New York, NY, USA, 2016. Association for Computing Machinery.

[21] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella. PANIC: A High-Performance programmable NIC for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 243–259. USENIX Association, Nov. 2020.

[22] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 318–333, New York, NY, USA, 2019. Association for Computing Machinery.

[23] MathWorks. HDL Coder - Generate Verilog, SystemVerilog, and VHDL code for FPGA and ASIC designs. https://www.mathworks.com/products/hdl-coder.html, 2025.

[24] Microchip Technology Inc. MIV RV32. https://www.microchip.com/en-us/products/fpgas-and-plds/ip-core-tools/miv-rv32. Accessed: 2025-07-08.

[25] Microchip Technology Inc. AN4568: PolarFire SmartSFP Plus Solution Featuring In-Application Programming. Technical Report AN4568, Microchip Technology Inc., 2022. Accessed: 2025-07-08.

[26] Microchip Technology Inc. AN4364: PolarFire FPGA SFP+ Module. Technical Report AN4364, Microchip Technology Inc., 2023. Accessed: 2025-07-08.

[27] Microchip Technology Inc. MPF200T PolarFire FPGA. https://www.microchip.com/en-us/product/mpf200t, 2025. Accessed: 2025-07-08.

[28] J. Min, M. Liu, T. Chugh, C. Zhao, A. Wei, I. H. Doh, and A. Krishnamurthy. Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 106–122, New York, NY, USA, 2021. Association for Computing Machinery.

[29] NVIDIA Corporation. Introduction to NVIDIA BlueField-2 InfiniBand/Ethernet DPU. https://docs.nvidia.com/networking/display/bluefield2dpuvpi/introduction, 2023. Accessed: 2025-07-08.

[30] OSFP MSA Group. OSFP (OCTAL SMALL FORM FACTOR PLUGGABLE) MULTI-SOURCE AGREEMENT ("MSA"). https://osfpmsa.org/assets/pdf/OSFP_MSA_20211004.pdf, 2021. Accessed: 2025-10-13.

[31] PlumSpace. Smart SFP. https://plumspace.com/products/smart-sfp/, 2025. Accessed: 2025-07-08.

[32] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, et al. Flowblaze: Stateful packet processing in hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 531–548, 2019.

[33] QNAP Systems, Inc. QNA-T310G1S. https://www.qnap.com/en-us/product/qna-t310g1s, 2025. Accessed: 2025-07-08.

[34] QSFP-DD. QSFP-DD MSA Hardware Rev 7.1. http://www.qsfp-dd.com/wp-content/uploads/2024/07/QSFP-DD-Hardware-Rev7.1.pdf, 2024. Accessed: 2025-10-13.

[35] QSFP112 MSA. QSFP112 Published Specification Rev. 2.1.2. http://www.qsfp112.com/QSFP112_MSA_Specification_Rev2.1.2.pdf, 2025. Accessed: 2025-10-13.

[36] QSFPTEK. Generic SFP-10G-SR 10GBASE-SR SFP+ Transceiver Module. https://www.qsfptek.com/product/100390.html, 2025. Accessed: 2025-07-08.

[37] R. Pérez-Aranda. VCSEL reliability analysis for technical feasibility assessment. https://www.ieee802.org/3/OMEGA/public/nov_2019/perezaranda_OMEGA_05a_1119_VCSEL_Reliability.pdf, 2019.

[38] A. Rivitti, R. Bifulco, A. Tulumello, M. Bonola, and S. Pontarelli. ehdl: Turning ebpf/xdp programs into hardware designs for the nic. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 208–223, New York, NY, USA, 2023. Association for Computing Machinery.

[39] H. Sadok, A. Panda, and J. Sherry. Of apples and oranges: Fair comparisons in heterogenous systems evaluation. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, page 1–8, New York, NY, USA, 2023. Association for Computing Machinery.

[40] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik. Scaling distributed machine learning with In-Network aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 785–808. USENIX Association, Apr. 2021.

[41] SHI International Corp. NVIDIA BlueField-2 SmartNIC P-Series DPU MBF2H332A-AECOT. https://www.shi.com/product/44041656/NVIDIA-BlueField-2-SmartNIC-P-Series-DPU-MBF2H332A-AECOT, 2025. Accessed: 2025-07-08.

[42] SHI International Corp. Pensando Distributed Services Platform DSC-25 Card. https://www.shi.com/product/40341946/Pensando-Distributed-Services-Platform-DSC-25-Card, 2025. Accessed: 2025-07-08.

[43] V. Shrivastav. Stateful multi-pipelined programmable switches. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 663–676, New York, NY, USA, 2022. Association for Computing Machinery.

[44] A. Sivaraman, M. Budiu, A. Cheung, C. Kim, S. Licking, G. Varghese, H. Balakrishnan, M. Alizadeh, and N. McKeown. Packet transactions: High-level programming for line-rate switches, 2015.

[45] SmartSFP. SmartSFP products overview. https://www.smartsfp.com/product-overview. Accessed: 2025-07-08.

[46] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li. In-band network telemetry: A survey. *Computer Networks*, 186:107763, 2021.

[47] Zephyr. Zephyr RTOS. https://www.zephyrproject.org/, 2025. Accessed: 2025-07-08.

[48] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1083–1100. USENIX Association, Nov. 2020.

[49] C. Zheng, M. Zang, X. Hong, L. Perreault, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman. Planter: Rapid prototyping of in-network machine learning inference. *SIGCOMM Comput. Commun. Rev.*, 54(1):2–21, Aug. 2024.