OpenDesc: From Static NIC Descriptors to Evolvable Metadata Interfaces

Seyyidahmed Lahmer
UCLouvain

Nikita Tyunyayev
UCLouvain

Tom Barbette
UCLouvain

Abstract

Modern NICs offer rich functionalities, but host-side software lacks a unified way to express or adapt to their capabilities. Instead, developers rely on device-specific code and ad-hoc glue layers, leading to duplication, reduction to the lowest common denominator, inefficiency, and poor reuse. As NICs grow more flexible, the lack of a shared interface description becomes a core architectural bottleneck.

We propose **OpenDesc**, a common description interface based on P4. While P4 is typically used to define the data plane, we additionally repurpose it to allow the NIC and host to describe their roles in packet exchange.

The host declares an intent, specifying what functionalities should be implemented by the NIC. Fixed-function NICs describe what functionalities they can provide and their interface. Programmable NICs describe the constraints of their interface. OpenDesc uses the combinations of intents and capabilities to select a NIC-compatible descriptor format and compile a NIC-specific driver code that is tailored to the application's intent. Missing features are implemented in software, or pushed to the programmable pipeline if available.

We showcase a prototype OpenDesc compiler that can select the fittest interface from the NIC interface description, and generate the associated host code for multiple NICs. The OpenDesc prototype enables access to the metadata sent from the NIC in eBPF through XDP or userlevel programs directly accessing the NIC descriptors. Our compiler is a first step towards enabling a generated minimalist driver datapath that can leverage the growing capabilities of increasingly feature-rich NICs.

CCS Concepts

• **Networks** → **Network adapters**; **Programmable networks**; • **Software and its engineering** → Source code generation.

Keywords

SmartNIC, Offload, P4, Metadata Interface

If you cite this paper, please use the ACM HotNets reference: Seyyidahmed Lahmer, Nikita Tyunyayev, and Tom Barbette. OpenDesc: From Static NIC Descriptors to Evolvable Metadata Interfaces. The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA https://doi.org/10.1145/3772356.3772418 .

1 Introduction

Modern network interfaces have outgrown their original role. Once simple *Dumb DMA* engines, NICs now carry out a wide range of tasks: checksum verification, flow hashing, timestamping, encryption, tunneling, even limited forms of scheduling and stateful processing. At the same time, host-side software has grown more dependent on these capabilities to save CPU cycles and increase throughput [3, 16, 20, 24, 27, 39].

In a typical system, the host and NIC coordinate using descriptor queues: structured memory regions shared via Direct Memory Access (DMA). These descriptors control how packets are sent and received, and they encode metadata, such as timestamps, hashes, protocol state, offload flags, and more. However, each vendor defines this metadata differently. Even the presence or absence of a field is vendor-specific, and the layout may change with firmware updates, product revisions, or the addition of new features [2].

This problem will only get worse in the future. Programmable NICs can implement new parsing logic [32], configure match-action pipelines [4, 19, 30, 50], and insert custom metadata [53]. This raises two challenges. First, custom new features often introduce a need to exchange metadata with packets, such as cryptographic context for AES offload [21] and more generally L5 offloads [38]. Dedicated queues for off-path offloads can even be used to exchange (partial) packets with specific metadata, for instance for RegEx offload [29], or chunk of payloads instead of packets with TCP offload [22, 28]. What is exchanged with the SmartNIC accelerators often reduces to a pointer, a length, plus some specific accelerator-dependent metadata. It is not possible for a single interface to match all those use-cases. Second, the behavior now depends on the program currently running on the NIC. One device might, in one configuration, generate a timestamp and a flow hash; in another, it might expect the host to supply steering hints or scheduling directives. A single NIC becomes both a producer and consumer of metadata, depending entirely on the deployed data plane.

Exposing the ever-growing capabilities of modern NICs to software remains fragmented and ad hoc. Kernel stacks like Linux' one use heavyweight abstractions such as `sk_buff` to store metadata, which hurts performance [6, 42, 47]. XDP [16] improves efficiency by narrowing the interface, but supports only a limited subset of metadata, and requires manual coordination between drivers and applications for additional

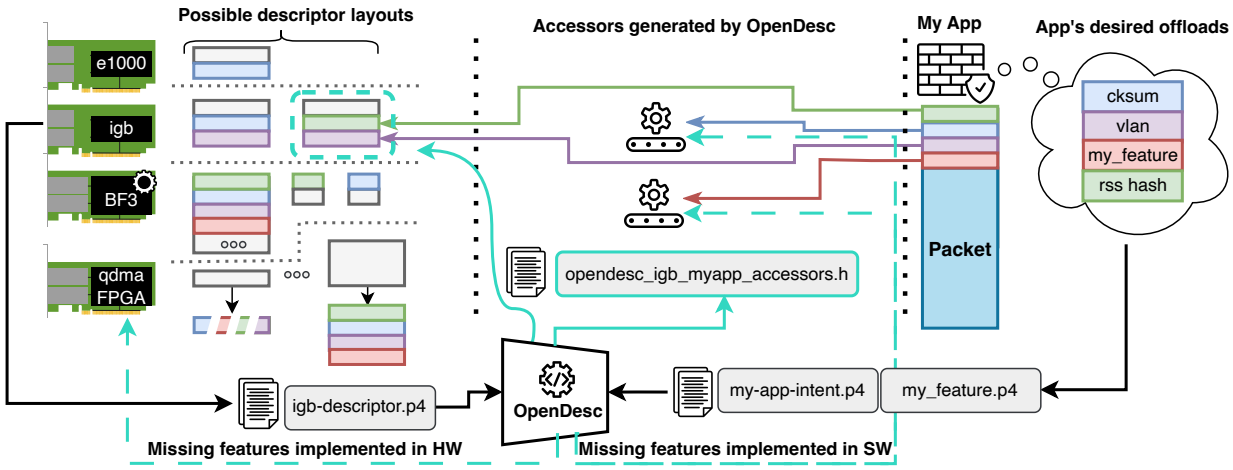


Figure 1: OpenDesc is the bridge between various descriptor formats of fixed and programmable NICs, and various application and drivers layouts

fields. Embedding drivers in applications through a clean-slate approach can offer performance benefits [6, 7, 35, 41], but is difficult to maintain and scale to multiple NICs. Other works moved the descriptor data inside the packet itself [47], reimplemented them in software pipelines [14], or just removed them [43]. This highlights the tension between standardization and performance, but ultimately falls short of offering a unified, dynamic metadata handling mechanism across devices.

We present OpenDesc, a model and implementation to describe the NIC-host communication semantics using P4, extended with some code annotations. P4 is used as a declarative interface contract. This contract describes metadata fields, types, semantics, and directionality. With it, software frameworks can auto-generate parser code, validate NIC behavior, and negotiate features. NICs become self-describing; host applications become portable. The result is a network I/O path that is simpler, more predictable, and more future-proof.

We further extend on the motivation behind our work and previous work in Section 2. We describe the interface in Section 3. We propose a prototype compiler that transforms a given NIC interface description and application intent into a descriptor layout and application accessor in Section 4. We conclude with open research areas in Section 5.

2 Why OpenDesc?

To cope with diversity in NICs' descriptors and features, each software stack writes its own interpretation logic. Linux drivers extract nearly all possible metadata into large `sk_buff` structure. As this interface is slow [6, 42, 43, 47], XDP [48] was introduced and provide a least-common-denominator through `xdp_buff` fields. However, XDP descriptors do not contain any offload information, such as the packet's timestamp or RSS hash. XDP, therefore, proposes 3 accessors (at the time of writing), which are defined by a few drivers, to fetch the supplementary data from the NIC descriptor if the compiled eBPF program accesses it. Every new offload requires coordination between firmware teams, driver authors, framework maintainers, and application developers. For instance, the BPF accessors only cover 3 of the 12 metadata information available in NVIDIA Mellanox ConnectX descriptors. Kernel bypass solutions like Netmap [42] choose the path of least-common-denominator, a buffer pointer and the packet length. DPDK [25] has per-driver code, setting some of the 128bytes DPDK descriptor space `rte_mbuf`. Although the `rte_mbuf` structure is already quite large, it eventually became insufficient to store the growing number of offloads [8]. DPDK, therefore, introduced an indirection layer that copies metadata based on numerous configuration flags [9], a mechanism that has itself become a performance bottleneck [35, 47].

This process consumes engineering time. It ossifies innovation: even if a NIC supports a new offload, it is only useable if all the relevant software is updated to match. Worse, it fragments the ecosystem. There is no standard way for a host to ask, "What metadata can you give me for this packet?" Nor for a NIC to declare, "Here is the structure and meaning of what I will write." Even the XDP accessor model cannot cope with dynamic pipelines, as its static accessors would need to be defined according to the programmable pipeline currently running on the programmable NIC.

Observing the slowness introduced by the jungle of metadata in DPDK, TinyNF [35] observed a 1.7x throughput improvement by simply re-implementing the driver for a single NIC. Similarly applications like VPP [7], Caladan [10] or Cornflakes [41] moved away from DPDK and implemented direct userlevel support for the NVIDIA ConnectX NICs, because the datapath of DPDK became too slow for them. Cornflakes particularly highlights the problem. To offload protobuf-like serialization, Cornflakes needs to write many TX descriptor entries to benefit from the scatter/gather support of the NIC, offloading serialization. Allocating many DPDK structures proved too slow just to write a series of lengths and offsets in the TX ring, leading authors to re-implement a "mini-DPDK" [41]. This is, of course, not a practical solution as it needs a lot of engineering effort to stay compatible with every firmware update, and scale to more devices. X-Change [6] collapsed DPDK's datapath driver and DPDK application into a unified runtime using link-time optimization (LTO). This reduces transformation overhead, leading to an increase of 70% for the throughput and a reduction of 28% for the latency. X-Change requires a complete rewrite of the driver datapath code and remains limited to fixed functions. Most recently, ASNI [47] proposed aggregating packets and their descriptors within a larger packet, known as an ASNI frame. ASNI, therefore, circumvents the problem by embedding metadata within the packet buffer itself. Still, ASNI only works with programmable NIC and does not allow to dynamically find out if a feature is available or not as the layout of the descriptor is fixed, only enabling or disabling some fields. ENSO [43] replaces descriptor rings with a streaming model. This led to a 6x throughput improvement for raw payload processing, but does not enable the exchange of packet metadata with the NIC. Therefore, the model collapses if the application needs to recompute metadata such as a hash in software [47]. Furthermore, by removing the indirection provided by the descriptors, ENSO does not allow packets to be processed in a different order from the one in which they were received without copying them. SoftNIC [14] addresses the problem by introducing a software-emulated NIC as a middle layer, implementing complex functions (e.g., parsing, shaping, classification) in software pipelines.

Following the guidelines of these previous works would result in reimplementing support for each NIC in every application. Essentially, this is equivalent to reimplementing DPDK drivers, but with a distinct datapath for each application. SoftNIC highlights this tension: While it unifies application-facing APIs, it requires developers to define detailed software pipelines per target NIC to fill in missing functionality – without any standardized way for the NIC to describe what it does or does not provide.

In OpenDesc, we propose to generate the variable portion of the datapath according to the specification of the NIC, and the needs of the application. The rest – which is a vast majority – of the drivers' code remains intact.

Figure 1 shows an overview of OpenDesc. We consider an application that wants to receive the checksum of a packet, the decapsulated vlan TCI, the RSS hash and the result of a specific feature, for instance the key of a key-value-store request as done in previous work [20]. Each NIC comes with a P4 definition of its descriptor parser and completion serializer. Fixed-function NICs declare the supported layouts; older NICs like the early Intel e1000 series supported only a single descriptor, giving the computed IP checksum of the packet. Newer Intel NICs came with a bigger descriptor that can contain the RSS hash, or the checksum, but not both. One of the layouts must be chosen; the other 3 features must be recomputed in software. Some partially programmable NICs like the NVIDIA BlueField 3 enable a field for specific metadata computed through a series of Match-Action tables, recently programmable in P4 [30]. However, the complete descriptor that can potentially fit the 4 offloads is large. One might prefer to use the compressed descriptor format that the NIC also supports, which might contain only the hash, or only the checksum. Finally, fully programmable NICs, like those using the Xilinx QDMA API [1], have fully programmable descriptors of 8, 16, 32 or 64 bytes.

The OpenDesc compiler infers what fields the NIC can provide, synthesizes parser and accessor logic, and aligns host and NIC capabilities automatically. This decouples applications from specific NIC quirks. We propose each offload feature to come with a reference P4 implementation. If hardware lacks capability, OpenDesc can delegate to software (e.g., a SoftNIC-like augmentation) using P4-to-software compilers [5, 44, 49]. For programmable NICs, missing features can therefore be pushed to the NIC using one of the numerous P4-to-device compilers [19, 30, 45, 50, 52]. We note our current prototype compiler only lists the missing features for the given NIC, and the one left out after considering all possible layouts (more in §4), but does not currently offload or compile the P4 code to software.

OpenDesc builds on a long line of work on SmartNICs, but provides a missing cornerstone: explicit, declarative *descriptor semantics* that bridge the gap between hardware capabilities and software needs.

Other prior systems than the one mentioned above have tackled parts of the problem, often by moving functionality rather than defining a shared language. Even the emerging P4-based NIC architecture efforts leave descriptor semantics open. The P4 Portable NIC Architecture (PNA) [32] explicitly recognizes that NICs need on-card message processing for DMA, segmentation, etc., but defers all details of descriptor formats to the NIC vendor. FlexNIC [20] offloads some host processing into the NIC itself, allowing applications to install

simple match-action rules on the NIC hardware. Both PNA and FlexNIC give no prescription for how a host should interpret NIC metadata fields and how to put both on the same page.

Clara [40] and Alkali [23] explore related gaps: Clara builds a performance model of how network functions use NIC resources, and Alkali provides an intermediate representation to target diverse NIC hardware. We explore how performance interfaces of packet operations can be used to tailor the choice in OpenDesc in Section 5.

In summary, past work has addressed programmability, performance, and even interface alternatives, yet all rely on implicit, hard-coded metadata interpretations. OpenDesc bridges this gap by introducing a *descriptor-level semantic contract*.

3 OpenDesc interface

What a NIC produces or consumes should not be inferred from vendor documentation, reverse engineering, or constant trial-and-error integration. It should be declared in a language understood by both ends of the link. A compiler can act as a mediator: map what the host wants to consume to what the NIC can produce, based not on identical formats but on shared structure and intent. The goal is not uniformity, but semantic alignment.

P4 is typically used to describe how programmable data planes parse packets and apply match-action logic. In our setting, we repurpose P4 not only to define forwarding behavior but also to describe interface semantics, which means specifying what metadata is exchanged between the host and NIC, where it appears, and how it should be interpreted. We select P4 as it is a domain-specific language designed solely for describing the data plane behavior of network devices. It also proposes a suitable bit-wise header description format, at the heart of OpenDesc. While some alternatives, such as eBPF, could be appropriate, they are often too expressive, which complicates the analysis of the parser code. We elaborate on the tradeoffs of P4 in Section 5.

Each direction of the packet (TX and RX) involves a producer and a consumer. In RX, the NIC produces metadata such as flow hashes, timestamps, and checksum status; the host consumes it. On TX, the host produces intent – such as requesting segmentation, tagging, or offload parameters – and the NIC consumes it. These metadata structures vary by pipeline, firmware, and device capability and must be described explicitly.

Each direction, TX and RX, involves a producer and a consumer of metadata. On TX, the host sets the packet to be transmitted along with some offload hints. On RX, the NIC DMAes the received packet along with some precomputed results such as flow hashes and timestamps. These

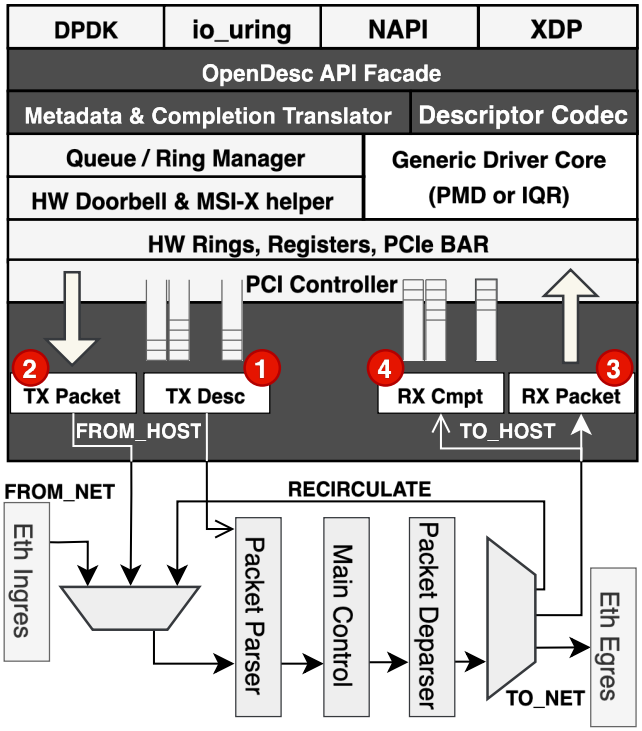


Figure 2: OpenDesc Architecture.

interactions can be categorized into five channels, as shown in Figure 2:

- **TX Desc** 1 **:** Transmit descriptors posted by the host, containing buffer addresses, lengths, and offload flags or parameters.
- **TX Packet** 2 **:** The packet payload itself, which the NIC's DMA engine reads from host memory to process/transmit onboard.
- **RX Packet** 3 **:** The packet payload received from the wire, which the NIC's DMA engine writes into host memory.
- **RX Cmpt** 4 **:** Completion records written by the NIC back to the host, indicating the status of a received packet and containing computed metadata such as flow hashes or timestamps.
- **Control Channel (Implicit):** Configuration and control messages, typically handled out-of-band via mechanisms like MMIO writes to hardware registers

Packet data follows a predefined path (e.g., following PNA) through a pipeline, with a (perhaps programmable) parser, control, and deparser. But descriptor metadata requires additional machinery. We introduce two additional blocks, positioned logically outside the packet path, which describe how descriptors and completions are interpreted or emitted:

- **Descriptor Parser:** interprets the host's posted TX

descriptor. This typically involves raw memory mapped through DMA and converted into structured fields. It may depend on the context of the queue or the ring layout.

- **Completion Deparser:** emits metadata from NIC to host in the form of RX completions. This includes computed hashes, timestamps, and offload results.

In OpenDesc, these roles are explicit as part of the pipeline description. The relevant components appear as follows.

The *Descriptor Parser* shown in Figure 3 is a P4 parser element that interprets a stream of bytes (of type `desc_in`) using some context information–such as per-queue layout or descriptor size–to guide the parsing logic. It outputs a structured collection of header fields that can be subsequently accessed by the main packet processing pipeline or by the completion logic (discussed next). The parser is designed as a templated component, allowing each NIC implementation to define its own descriptor parsing strategy tailored to its specific layout and semantics.

The *Completion Deparser* shown in Figure 4 abstracts the completion channel, which is DMA-ed back to the host to convey both the status of the received packet and the associated computed metadata. This control block is placed at the final stage of the pipeline, where it has access to both the parsed descriptor (that is, the output of the `DescParser`) and the metadata produced by the pipeline logic itself (represented by the `pipe_meta` parameter). Its role is to serialize the relevant fields from these inputs into a well-defined byte stream that is emitted to host memory. For example, if the pipeline computes a 5-tuple flow hash, the `pipe_meta` piggybacks this value, and then the `CmptDeparser` ensures that this value is emitted at the correct offset within the completion structure, aligned with the expectations of the host-side facade API. This alignment ensures proper interception and interpretation of metadata by the application.

The application declares its intent metadata as a simple P4 `header` type as shown in Figure 5. Certain fields may or may not be present depending on the NIC capabilities. We use `@semantic` annotations to tag each field with a semantic name that corresponds to a native type recognized by OpenDesc. The application can define new `@semantic` annotations that are tied, for instance, to a new feature that will be offloaded in a programmable NIC or future NICs.

We note applications might use multiple OpenDesc instance with different intents to obtain different queues tailored for different kind of traffic. This is orthogonal to the declarative interface itself.

4 Compiler Mapping

We implement an OpenDesc prototype compiler that bridges a NIC's metadata behavior and an application's declared intent. The prototype is implemented in C++, building on the p4c frontend for parsing and IR analysis. The compiler currently showcases how the declarative interfaces can be used to select the best descriptors supported by the NIC and generate accessors to the descriptor for an application.

In what follows, we describe the key steps of the process.

1. Control-flow graph extraction. The compiler parses the body of CmptDeparser once, replacing each `emit` statement by a vertex and each conditional by two directed edges labeled with the branch predicate that guards them. A root-to-leaf walk in this graph $G = (V, E)$ is called *completion path*, forming a concrete metadata layout that the NIC may emit under a given context. Each vertex $v \in V$ carries three static properties that can be determined by symbolic evaluation. Figure 6 illustrates this process: the right-hand side shows the original P4 logic of a simplified `CmptDeparser` for the Intel `e1000` newer NICs. The left-hand side depicts its corresponding control-flow graph, in which the branch on `ctx.use_rss` (i.e., a config variable) selects between emitting a single RSS 32bits hash or two alternative metadata fields (`ip_id` and `csum`). Each node in the graph corresponds to a static `emit` call and carries three properties used for analysis: the emitted bit range, the associated semantic meaning (e.g., `rss`, `ip_checksum`), and its byte size.

$\text{bits}(v)$	contiguous byte range committed by v
$\text{sem}(v)$	subset of Σ that those bytes encode
$\text{size}(v)$	$\lvert\text{bits}(v)\rvert$ in bytes

2. Path characterization. For a path $p = (v_0, \ldots, v_k)$ let

$$\text{Prov}(p) = \bigcup_{i=0}^{k} \text{sem}(v_i), \qquad \text{Size}(p) = \sum_{i=0}^{k} \text{size}(v_i).$$

Applications request the set $\text{Req} \subseteq \Sigma$ that appears in their intent header. Every semantic $s \in \text{Req} \setminus \text{Prov}(p)$ must be computed in software. A SoftNIC-like framework emulates each missing semantic at a run-time cost: $w : \Sigma \to \mathbb{R}_{>0} \cup \{\infty\}$.

3. Optimization problem. The compiler chooses a completion path by solving the following optimization.

$$\min_{p \in \text{Paths}(G)} \Big(\underbrace{\sum_{s \in \text{Req} \setminus \text{Prov}(p)} w(s)}_{\text{SoftNIC cost}} \quad \underbrace{\text{Size}(p)}_{\text{DMA completion footprint}} \Big) \tag{1}$$

The DMA completion footprint is a positive real value proportional to the size of the completion descriptor for the path p, and favors paths with smaller completions.

The first element minimizes CPU overhead; the second one favors shorter descriptors, which reduce DMA bandwidth. If

```
parser DescParser<
 typename H2C_CTX_T, typename DESC_T>(
    desc_in          desc_in,
    in   H2C_CTX_T h2c_ctx,
    out  DESC_T    desc_hdr
);
```

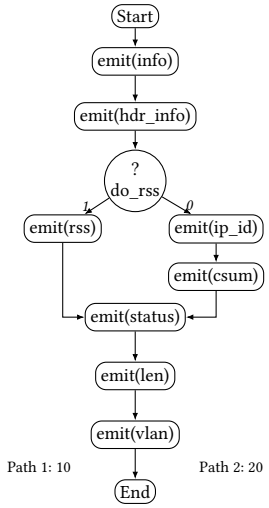
Figure 3: Descriptor parser template

```
control CmptDeparser<
  typename C2H_CTX_T,
  typename DESC_T,
  typename META_T>(
    cmpt_out       cmpt_out,
    in  DESC_T     desc_hdr,
    in  META_T     pipe_meta
);
```

Figure 4: Completion deparser template

```
header intent_t{
  @semantic("rss")
  bit<32> rss_val;
  @semantic("vlan")
  bit<16> vlan_tag;
  @semantic("ip_checksum")
  bit<16> csum;
}
```

Figure 5: Semantic annotations




```
@card2host
control CmptDeparser(
    cmpt_out out,
    c2h_ctx_t  ctx,
    in  h2c_desc_t   dh,
    in  meta_t   m) {
    apply {
        out.emit(m.pkt_info);
        out.emit(m.hdr_info);
        if (ctx.use_rss == 1) {
            out.emit(m.rss_hash);
        } else {
            out.emit(m.ip_id);
            out.emit(m.csum);
        }
        out.emit(m.status_error);
        out.emit(m.length);
        out.emit(m.vlan);
    }
}
```

Figure 6: Simple completion serializer for `e1000`. Left: control flow graph of the serializer program. Right: P4 logic of the `CmptDeparser` that emits completion fields based on a single bit context.

the first element is ∞ for every path, the program is rejected as unsatisfiable.

Because production NICs expose only a handful of completion paths (two in `e1000`, many formats for MLX5, one per installed queue in fully-programmable cards like those based on QDMA), optimization degenerates into enumerating a small finite set and picking the best element of (1). Figure 6 provides the running example: when both `rss` and `csum` are requested, the compiler prefers the right branch because it is assumed that the software rss is cheaper than recomputing the csum (using the cost given for the annotated feature).

4. Host stub synthesis. After selecting an optimal path $p^\star$ the compiler emits:

- *Constant-time accessors* for every $s \in \mathrm{Prov}(p^\star)$ that read the relevant bit slice directly.
- *SoftNIC shims* for the remaining semantics. Each shim is a thin wrapper around a library routine that implements s. Currently, the user is informed of missing s and is responsible for providing a linkable software implementation. We envision to automatically compile the routine from a given P4 reference implementation using existing compilers [31, 49].

The metadata can then be accessed using the generated accessor functions that read a fixed offsets relative to the beginning of the descriptor. Using XDP, access to the descriptor can be bounded and therefore read safely from an eBPF program. In future work, we want to enable the use of the accessors in DPDK by enabling a hook on the descriptor, much like XDP is doing for kernel drivers. The accessors can also be used by applications directly manipulating the NIC, allowing to adapt to new offloads and firmware upgrades.

5 Discussion and future work

Our work establishes a foundation for portable, semantics-aware NIC–host interfaces. Although our current prototype focuses on completion metadata extraction, broader challenges remain in describing the full interface, reasoning about offload trade-offs, and targeting diverse hardware. In what follows, we outline several directions to extend OpenDesc's scope and impact.

Synthesizing the complete driver datapath Future work is required in order to express the dynamics of the interface atop the limited scope of the descriptors' content. An application could use batched descriptors, as ASNI [47] proposes, or a streaming interface, as Enso [43] introduces, according to the application's potential for optimal performance on a given NIC. Enabling a description of the interface semantics, such as how the device moves from one descriptor to the next, would enable generating the full datapath driver code.

Feature equivalence In OpenDesc, we propose the use of annotations that define the functionality desired by the application. This choice posits that a clear definition of all possible features should be provided so vendors and applications can agree on the semantics of the metadata header. A reference implementation should also be shipped for each functionality to be synthesized in software, or the programmable hardware if available. Several research efforts focused on compiling P4 code to the specifics of SmartNICs [11, 12, 46, 54, 55], showing that P4 is suitable to describe NIC functionalities. Future work could enable the ability to use symbolic execution, as done in previous work on network functions and NIC

code [33, 36, 37], to understand if a feature described in the NIC is equivalent to a feature described in application code and avoid the need for standardization of functionalities.

Stateful offloads OpenDesc does not currently implement stateful features, though in principle they could be described using P4 primitives such as registers or externs. Since these constructs are used only as a descriptive mechanism and are not mapped to hardware resources, their number or size is not a constraint. Nonetheless, a more generic and extensible abstraction will be needed to express such stateful behaviors in a portable way across NICs.

Limits of P4 for defining complex offloads P4's restrictions also come with limitations in what can be expressed to define complex offloads. Certain metadata extends beyond per-header information, requiring context that spans multiple packets or even depends on inspecting payload bytes, such as pattern-matching. However, we still believe P4 is suitable for the *interface* itself. While the idea that all features could be *entirely* defined in a unique language is undoubtedly appealing, the application demands have to match the vendors' definitions. If both sides describe a very complex offload, the slightest difference will conclude that the feature is not available in the NIC. Even for something relatively simple, such as the RSS hash, we realized a symbolic analysis approach wasn't going to work as implementations from vendors differ slightly and offer different hashing schemes. However, the user only wants a mash-up of bits that is consistent per-connection and as different as possible between connections. The implementation details are less relevant for the user. This is why, in the end, we resorted to using annotations. P4, however, enables access to more complex offloads through *extern*, special externally implemented objects or functions that can be invoked from P4 code. Future work could tie externs to a given implementation, using an appropriate language for a complex feature. There is, however, no need for the interface to be able to peak in the feature itself. Multiple previous works such as Floem [34], iPipe[26], or Alkali [23] enable some languages to define functionalities that can be compiled to different targets. While these compilers could be used to define some complex functionalities, more research is needed to assert if the opposite is true: whether the limits of each target (FPGA, ASICs, muti-core NICs, ...) can be expressed using a unique language.

Performance and programmable constraint Not all features might be actionable at the same time, and programmable NICs offer constrained resources. Given all applications' intent, an open question is still whether a feature should be offloaded to the NIC even if technically possible, or if sometimes using a software counterpart is not more desirable. On the NIC side, Pipeleon [55] computes a performance cost for a given P4 program for multiple SmartNICs. Iyer et al. [18] advocate for a clear NIC performance interface. LogNIC [13] proposes a packet-centric approach to model the performance of a SmartNIC and its accelerators. P4All [15] proposes elastic structures in P4 programs to dynamically size storage requirements when programming P4 pipelines. On the host side, PIX [17] enables building a performance model of network functions. OpenDesc presents the opportunity to combine these works for a more opportunistic placement beyond the question of whether a feature is available or not. Similarly, Menshen [51] tackles the issues of isolation when concurrent pipelines run in a programmable RMT architecture, which could be informed by the application's intents to provide isolation inside NICs.

SIMD and architecture-dependent optimization Most DPDK drivers implement another version of the driver datapath using SSE (128-bit) to read 4 descriptors at a time, and one for different architecture (Altivec or Neon). This pose a burden on maintainance. OpenDesc could be extended to generate SIMD accessors instead, taking also advantage of wider lanes to process more packets in parallel.

Acknowledgments

We thank our shepherd, Gábor Rétvári, for his guidance during the preparation of the camera-ready version, as well as the anonymous reviewers for their constructive feedback. This work was supported in part by the UCLouvain FSR and the *Fonds de la Recherche Scientifique* (FNRS) through the MIS grant 40020886.

References

[1] AMD, Inc. 2025. *QDMA Subsystem for PCI Express Product Guide (PG302), Version 5.1.* AMD, Inc. https://docs.amd.com/r/en-US/pg302-qdma Accessed: July 1, 2025.

[2] Matan Azrad. 2019. [dpdk-dev] [PATCH v2 25/28] net/mlx5: handle LRO packets in Rx queue scheduling. https://mails.dpdk.org/archives/dev/2019-July/139167.html. DPDK Development Mailing List.

[3] Tom Barbette, Georgios P. Katsikas, Gerald Q. Maguire, and Dejan Kostić. 2019. RSS++: load and state-aware receive side scaling. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (Orlando, Florida) *(CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 318–333. doi:10.1145/3359989.3365412

[4] 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. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 99–110. doi:10.1145/2534169.2486011

[5] Tharun Kumar Dangeti, Ramakrishna Upadrasta, et al. 2018. P4llvm: An llvm based p4 compiler. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 424–429.

[6] Alireza Farshin, Tom Barbette, Amir Roozbeh, Gerald Q Maguire Jr, and Dejan Kostić. 2021. PacketMill: toward per-Core 100-Gbps networking. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 1–17.

[7] FD.io Project. 2025. Vector Packet Processing (VPP) - Source Code. https://github.com/FDio/vpp. Accessed: July 9, 2025.

[8] Linux Foundation. 2024. *DPDK API documentation - mbuf*. Linux Foundation. https://doc.dpdk.org/api/structrte__mbuf.html

[9] Linux Foundation. 2025. *DPDK API documentation - mbuf_dyn*. Linux Foundation. https://doc.dpdk.org/api/rte__mbuf__dyn_8h.html#a1c9c942a72b8a958818cd16955a65628

[10] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating interference at microsecond timescales. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 281–297.

[11] Jiaqi Gao, Ennan Zhai, Hongqiang Harry Liu, Rui Miao, Yu Zhou, Bingchuan Tian, Chen Sun, Dennis Cai, Ming Zhang, and Minlan Yu. 2020. Lyra: A Cross-Platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs. 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* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 435–450. doi:10.1145/3387514.3405879

[12] Xiangyu Gao, Taegyun Kim, Michael D. Wong, Divya Raghunathan, Aatish Kishan Varma, Pravein Govindan Kannan, Anirudh Sivaraman, Srinivas Narayana, and Aarti Gupta. 2020. Switch Code Generation Using Program Synthesis. 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* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 44–61. doi:10.1145/3387514.3405852

[13] Zerui Guo, Jiaxin Lin, Yuebin Bai, Daehyeok Kim, Michael Swift, Aditya Akella, and Ming Liu. 2023. Lognic: A high-level performance model for smartnics. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 916–929.

[14] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. SoftNIC: A Software NIC to Augment Hardware. In *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155*.

[15] Mary Hogan, Shir Landau-Feibish, Mina Tahmasbi Arashloo, Jennifer Rexford, and David Walker. 2022. Modular switch programming under resource constraints. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 193–207.

[16] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. 2018. The eXpress data path: fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies* (Heraklion, Greece) *(CoNEXT '18)*. Association for Computing Machinery, New York, NY, USA, 54–66. https://doi.org/10.1145/3281411.3281443

[17] Rishabh Iyer, Katerina Argyraki, and George Candea. 2022. Performance Interfaces for Network Functions. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 567–584. https://www.usenix.org/conference/nsdi22/presentation/iyer

[18] Rishabh Iyer, Jiacheng Ma, Katerina Argyraki, George Candea, and Sylvia Ratnasamy. 2023. The case for performance interfaces for hardware accelerators. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*. 38–45.

[19] Jaco Joubert. 2018. Netronome P4 introduction at SIGCOMM'18. (2018).

[20] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 67–81. doi:10.1145/2872362.2872367

[21] Duckwoo Kim, SeungEon Lee, and KyoungSoo Park. 2020. A case for smartnic-accelerated private communication. In *Proceedings of the 4th Asia-Pacific Workshop on Networking*. 30–35.

[22] Taehyun Kim, Deondre Martin Ng, Junzhi Gong, Youngjin Kwon, Minlan Yu, and KyoungSoo Park. 2023. Rearchitecting the {TCP} stack for {I/O-Offloaded} content delivery. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 275–292.

[23] Jiaxin Lin, Zhiyuan Guo, Mihir Shah, Tao Ji, Yiying Zhang, Daehyeok Kim, and Aditya Akella. 2025. Enabling Portable and High-Performance SmartNIC Programs with Alkali. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 107–126. https://www.usenix.org/conference/nsdi25/presentation/lin-jiaxin

[24] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259. https://www.usenix.org/conference/osdi20/presentation/lin

[25] Linux Foundation. [n. d.]. Data Plane Development Kit (DPDK). https://www.dpdk.org. Accessed: 2025-07-10.

[26] Ming Liu, Tianyi Cui, Henrik Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. ipipe: A framework for building distributed applications on multicore soc smartnics. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[27] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Mike Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Mike Ryan, Erik Rubow, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a Microkernel Approach to Host Networking. In *In ACM SIGOPS 27th Symposium on Operating Systems Principles*. New York, NY, USA.

[28] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. {AccelTCP}: Accelerating network applications with stateful {TCP} offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 77–92.

[29] NVIDIA. [n. d.]. RegEx Programming Guide. https://docs.nvidia.com/doca/archive/doca-v1.4/regex-programming-guide/index.html

[30] NVIDIA. 2025. P4 Language Support in DPL. https://docs.nvidia.com/doca/sdk/P4+Language+Support+in+DPL/index.html

[31] P4.org. [n. d.]. BMv2. https://github.com/p4lang/behavioral-model. Accessed: 2025-10-01.

[32] P4.org. 2022. Portable NIC Architecture (PNA) Specification. https://p4.org/p4-spec/docs/PNA.html. Accessed: 2025-06-24.

[33] Francisco Pereira, Fernando M.V. Ramos, and Luis Pedrosa. 2024. Automatic Parallelization of Software Network Functions. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1531–1550. https://www.usenix.org/conference/nsdi24/presentation/pereira

[34] 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 *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 663–679.

[35] Solal Pirelli and George Candea. 2020. A Simpler and Faster NIC Driver Model for Network Functions. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX

Association, 225–241. https://www.usenix.org/conference/osdi20/presentation/pirelli

[36] Solal Pirelli and George Candea. 2020. A Simpler and Faster NIC Driver Model for Network Functions. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 225–241. https://www.usenix.org/conference/osdi20/presentation/pirelli

[37] Solal Pirelli, Akvilė Valentukonytė, Katerina Argyraki, and George Candea. 2022. Automated Verification of Network Function Binaries. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 585–600. https://www.usenix.org/conference/nsdi22/presentation/pirelli

[38] Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafrir. 2021. Autonomous nic offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 18–35.

[39] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. ZygOS: Achieving Low Tail Latency for Microsecond-scale Networked Tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 325–341. doi:10.1145/3132747.3132780

[40] Yiming Qiu, Qiao Kang, Ming Liu, and Ang Chen. 2020. Clara: Performance Clarity for SmartNIC Offloading. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (Virtual Event, USA) *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 16–22. doi:10.1145/3422604.3425929

[41] Deepti Raghavan, Shreya Ravi, Gina Yuan, Pratiksha Thaker, Sanjari Srivastava, Micah Murray, Pedro Henrique Penna, Amy Ousterhout, Philip Levis, Matei Zaharia, et al. 2023. Cornflakes: Zero-copy serialization for microsecond-scale networking. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 200–215.

[42] Luigi Rizzo. 2012. netmap: a novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*. 101–112.

[43] Hugo Sadok, Nirav Atre, Zhipeng Zhao, Daniel S. Berger, James C. Hoe, Aurojit Panda, Justine Sherry, and Ren Wang. 2023. Enso: A Streaming Interface for NIC-Application Communication. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 1005–1025. https://www.usenix.org/conference/osdi23/presentation/sadok

[44] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. 2016. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 525–538.

[45] Henning Stubbe. 2017. P4 compiler & interpreter: A survey. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM)* 47 (2017).

[46] Pensando Systems. 2023. Pensando DSC SmartNIC Compiler. https://p4.org/onf-product/pensando-dsc-p4-smartnic-compiler/.

[47] Nikita Tyunyayev, Clément Delzotti, Haggai Eran, and Tom Barbette. 2025. ASNI: Redefining the Interface Between SmartNICs and Applications. *Proc. ACM Netw.* 3, CoNEXT2, Article 9 (June 2025), 22 pages. doi:10.1145/3730966

[48] Marcos AM Vieira, Matheus S Castanho, Racyus DG Pacífico, Elerson RS Santos, Eduardo PM Câmara Júnior, and Luiz FM Vieira. 2020. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–36.

[49] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. 2018. T4p4s: A target-independent compiler for protocol-independent packet processors. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 1–8.

[50] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. 2017. P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*. 122–135.

[51] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. 2022. Isolation mechanisms for {High-Speed}{Packet-Processing} pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1289–1305.

[52] Loring Wirbel. 2014. Xilinx SDNet: a new way to specify network hardware. *Linley Group, Mountain View, CA, USA, Tech. Rep* (2014).

[53] AMD Xilinx. [n. d.]. Introduction • QDMA Subsystem for PCI Express Product Guide (PG302) • Reader • AMD Technical Information Portal. https://docs.amd.com/r/en-US/pg302-qdma/Introduction

[54] AMD Xilinx. 2023. Xilinx P4C SmartNIC (Alveo) and P4C FPGA (SDNet). https://p4.org/onf-product/xilinx-p4c-smartnic-alveo-and-p4c-fpga-sdnet/.

[55] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Songyuan Sui, Khalid Manaa, Omer Shabtai, Yonatan Piasetzky, Matty Kadosh, Arvind Krishnamurthy, TS Eugene Ng, et al. 2023. Unleashing SmartNIC packet processing performance in P4. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 1028–1042.