A Lightweight Southbound Interface for Standalone P4-NetFPGA SmartNICs

Mario Patetta, Stefano Secci, Sami Taktak

A Lightweight Southbound Interface for Standalone P4-NetFPGA SmartNICs

Mario Patetta, Stefano Secci, Sami Taktak
Cnam, Paris, France. Email: first-name.last-name@cnam.fr

***Abstract*—We present a lightweight Southbound Interface (SBI) for P4→NetFPGA devices, aimed at enhancing the capability of NetFPGA Smart Network Interface Cards (SmartNICs) to work in standalone mode. We propose a custom protocol allowing the control plane to remotely populate the board's routing tables and query runtime information. We achieve this thanks to the implementation of a lookup table (LUT) extern function that can be directly edited by the P4 program - a feature notably lacking in the context of P4→NetFPGA.**

***Index Terms*—P4, NetFPGA, Software-Defined Networking, Southbound Interface**

I. Introduction

Programmable switching hardware architectures provide flexibility in terms of resources allocation in packet processing, and the ability to implement custom functionalities. The integration of the software-driven approach allows easy deployment and update of novel designs.

P4 [1, 2] is the open source de facto standard programming language used to describe packet processing and forwarding in programmable network devices. P4 stands for *Programming Protocol-Independent Packet Processors* and is maintained by the P4 Language Consortium, a nonprofit entity hosted by the Open Networking Foundation.

P4 leverages two main abstractions. *Parsers* are finite state machines used to extract the headers out of the incoming stream of bytes, according to a predetermined flowchart. *Match-action tables* are then used to define how packet forwarding and processing is performed. Despite providing such specialized high level constructs for header manipulation, P4 lacks many others such as loops, recursion and pointers as well as stateful operations. To address these limitations, P4 offers the possibility to integrate some target-specific modules called *extern functions*, that can be interacted with as black boxes by P4 programs and that can implement additional functionalities.

The NetFPGA [3] is an open source hardware and software platform developed to provide a basic infrastructure to design, simulate and test networking platforms. The P4→NetFPGA project [4, 5] offers a development environment based on the Xilinx P4-SDNet toolchain [6]. The goal of this workflow is to allow developers to describe how packets are to be processed in the high-level P4 language, while still providing the possibility for experienced hardware designers to include custom extern functions described in Verilog or VHDL to the P4 pipeline.

Although the NetFPGA boards, as the SUME version, can be used in standalone mode, the P4→NetFPGA project relies on a host computer connected over PCIe to populate lookup tables [5]. In fact, P4-SDNet generates an Application Program Interface (API) that allows the host computer to manipulate registers and tables instantiated by the P4 program at runtime.

Allowing the boards to work autonomously as P4 switches would strongly enhance the flexibility of such platforms. To that purpose, we propose a basic SBI for P4→NetFPGA SmartNICs, through which the control plane can populate and update such tables, as well as retrieving runtime information from the board. For that purpose, we designed an extern module implementing a lookup table that can directly be edited by the P4 program. This enables the SmartNIC to be controlled through the SBI to interact with the P4 pipeline.

II. Background

In this section, we offer a glimpse on the related work in the scope of Southbound API and the use of SmartNICs to compute metrics directly on the data plane.

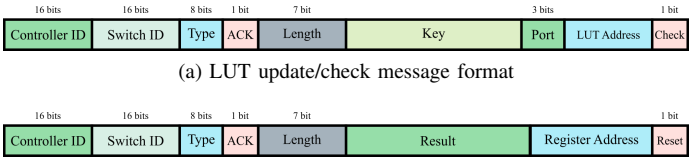
A. Metric computation in SDN Switches

The main advantage of using programmable packet processors is the capability to execute user-defined operations on top of forwarding. Some examples of tasks that can benefit of this are congestion control, flow monitoring, and intrusion detection. As opposed to centralized monitoring systems, data plane computing dramatically reduces the overhead due to information needing to be retrieved from switches as well as speeding up the process, since at least part of the computation can be performed at line rate.

An extensive and detailed survey on P4 applied research can be found in [7]. In [8–10] several heavy hitter detection algorithms are presented. BACON [11] and HashFlow [12] focus on flow analysis. BACON leverages the combination of data sketches for estimating flow cardinality, while HashFlow summarizes records of mice flows and keeps detailed ones for elephant flows in order to achieve a memory efficient yet accurate traffic analysis performance. PINT [13] uses P4 to add telemetry information into data packets with minimal overhead, while IntSight [14] leverages in-band network telemetry (INT) to monitor the network performance. P4Entropy [15] introduces two algorithms to bypass P4's lack of support for logarithm and exponential functions and uses them to estimate traffic entropy.

B. Southbound Interfaces

Some of the most widely used SBIs are P4Runtime [16], NETCONF [17], and OpenFlow [18].



(a) LUT update/check message format

(b) Metric query/reset message format

Fig. 1: SBI control-plane message formats

P4Runtime is growing in popularity in the field of P4 switches. Its syntax is defined in Protobuf [19], an open-source data format developed by Google that serves as a basis for gRPC [20], an open-source remote procedure call (RPC) library that runs on HTTP/2. P4 Runtime defines the communication between a gRPC server on the side of the switch and one or more gRPC clients on the side of the controllers. Similarly, also NETCONF uses an RPC paradigm. In this case, the data format used is XML [21]. The P4→NetFPGA project does not currently support any RPC server. It could be possible to build one running on the host computer, but this would require the platform not working in standalone mode.

OpenFlow is the most used and researched SBI. The main reason why it is generally not used in P4 networks is that, upon connecting, OpenFlow switches are supposed to actively initiate a TCP session using a three-way handshake with the SDN-Controller. However, regular P4 switches, including the SimpleSumeSwitch (i.e., the P4 architecture currently defined for the NetFPGA SUME), can only passively respond to packets. The Sume Event Switch, an experimental programmable architecture in the P4→NetFPGA project, can respond to multiple data plane events rather than just ingress packet arrival. This enhanced flexibility grants to the board the capability to generate packets on its own. Although exploring the use of the Sume Event Switch could be interesting for implementing OpenFlow as SBI for P4→NetFPGA switches, we chose to focus on a lightweight protocol that would be easier to manage and adapt to the specific behavior defined by the P4 program.

III. SBI: Control-Plane Message Formats

The SBI we propose is composed of four control-plane messages. Figure 1 illustrates the two control packet formats: LUT update/check (a) and metric query/reset (b).

The first 48 bits are in common to both the formats. *Controller ID* and *Switch ID* fields serve as a basic security purpose such that control packets with wrong ID pair will be dropped by the switch. *Type* field indicates whether the carried message is of type (a) or (b), while *ACK* is set to 0 for packets sent by the controller and to 1 for packets bounced back by the switch. Finally, The *Length* field expresses the length of the message in bytes: this value depends on the *Key* and *LUT address* fields for the LUT update/check message, and on the *Result* and *Register Address* fields for the Metric query/reset message.

A. LUT update/check message

The *Key* field in LUT update/check messages corresponds to the flow-rule for matching flows, while the *Port* field determines the port where to route packets destined to such flow. The *Address* field specifies the LUT address where to store the rule. Finally, the *Check* field is used to distinguish between LUT update and LUT check messages. In a controller to switch message, when *Check* is 1, *Key* is set to the flow-rule being checked (e.g. if the flow-rule is only based on destination IP address information, the destination IP address), while *Port* and *Address* are set to 0. Upon receiving such message, the switch fills the *Port* field with the output port corresponding to that Key and bounces the packet back to the controller.

In applications where multiple LUTs are used (i.e., if the flow-rule is based on multiple fields criteria), an additional *LUT ID* field may be added to discriminate them.

B. Metric query/reset message

In a metric query message, the controller sets the *Register Address* to read from, while the SmartNIC fills the *Result* field with the respective content. If the *Reset* field is set instead, the switch sets to 0 the content of the register at the specified address. These messages should be customized according to the application running on the SmartNIC, as number and size of the registers to query can vary as well as the bit size of the data they store. As in the case of LUT update/check messages, an optional *Register ID* field may be added.

IV. Data Plane Overview

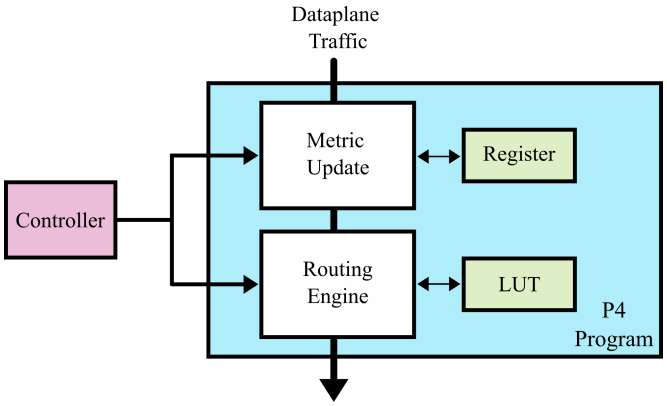


Fig. 2: Data Plane Overview

Figure 2 shows a P4 program running on a standalone NetFPGA SUME Board. The data plane traffic triggers the metric update and is routed according to one or more lookup tables. A remote controller can use the southbound interface to interact with both stages of the pipeline. LUT update/check messages are used to populate the lookup tables and to inspect their status. Metric query/reset messages are used to read the content of the registers or to clear it.

The remaining of this section describes the lookup table extern function we propose and the functionality of the routing engine.

A. Lookup Table Extern

```
#define LUT_READ        4w0
#define LUT_UPDATE      4w1

@CamLutKeyWidth(KEY_WD)
@CamLutAddressWidth(ADR_WD)
@CamLutNewValWidth(VAL_WD)
@Xilinx_MaxLatency(4)
@Xilinx_ControlWidth(0)
extern void <lut_name>_cam_lut(in  bit<KEY_WD> key,
                               in  bit<ADR_WD> address,
                               in  bit<VAL_WD> newVal,
                               in  bit<4>      opCode,
                               out bit<1>      match,
                               out bit<VAL_WD> result);
```

Fig. 3: LUT Extern Instantiation

Figure 3 shows the P4 instantiation of the exact match LUT extern function we designed. It is based on the Ternary Content Addressable Memory (TCAM) implementation in Verilog by mcjtag [22]. The memory consists of three components: the *line array*, which represents the actual Content Addressable Memory; the *line encoder*, which converts the output of the line array into a binary address for the RAM, that stores the data. The line array is used to match the input key to an address, that is used to read the corresponding data from the RAM.

The P4 pipeline interfaces with the extern through six I/O signals (i.e., extern metadata):

- *key*: when updating the LUT, it contains the new key to be written inside the line array. When reading the LUT, it is the data to look up for matches in the line array;
- *address*: it is the address inside both the line array and the RAM where the new rule will be written. It is set to 0 during the reading operation;
- *newVal*: when updating the LUT, it contains the new data to be written inside the RAM. It is set to 0 during the reading operation;
- *opCode*: it is set to 0 to read the LUT or to 1 to update it;
- *match*: it is set to 1 by the LUT when a reading operation results in a match;
- *result*: it is the value stored in the RAM at the address outputted by the line array if there is a match.

B. Routing Engine

The routing stage of the P4 pipeline is composed of two phases.

In the first phase, the packet is inspected in order to set the metadata for accessing the LUT extern. In case of data plane traffic, the flow-rule is set as *key* and the *opcode* is set to *READ*. In case of an update message, *key*, *address* and *port* are set like the respective fields and the *opcode* is set to *READ* or *UPDATE* according to the *check* field being 1 or 0.

In the second phase, the LUT extern is accessed. If the key produces a match, the result is used as destination port for IP packets or to fill the *port* field of check messages.

Control packets are always bounced back to the controller, i.e., the destination port is the same as the source port, while source and destination MAC addresses are swapped.

V. Technical Demonstration

In the technical demonstration, we showcase the operations of the P4→NetFPGA SBI we propose. For the sake of simplicity and without loss of generality, we only consider IP addresses as single matching field for flow rule definition.

We use TCP Monitor [23] as the P4 metric update application running on the SmartNIC; it is a basic TCP monitoring program provided by the P4→NetFPGA project. It computes the size of each flow that runs through the switch and, when the flow has ended (i.e., when a TCP FIN packet is received), it updates a histogram representing the distribution of flow sizes.

We modified the P4 code to adapt it to our metric query/reset messages, allowing us to read and reset the histogram register. The code and demo video are shared in [24]

For the demonstration, we show how our proposed SBI can be used by a remote controller to set the flow rules of a standalone SUME Board SmartNIC as well as retrieve runtime TCP monitoring metrics.

In the first part of the demo we populate the IP destination address LUT with LUT update messages and check the success of this operation by using LUT check messages.

Then, we send data plane traffic through the SmartNIC and use metric query messages to collect the monitoring information at runtime.

VI. Further Work

Even though instantiating lookup tables with hundreds or thousands of entries should be feasible in terms of memory footprint, it is not in terms of timing requirements. The size of the line encoder scales linearly with the number of entries, causing the presence of negative slack on the critical path. We are currently working on a parallel implementation of the line encoder in order to mitigate this phenomenon.

The SBI is currently using plain encapsulation over Ethernet frame, but this can be easily changed to upper layer protocols to favor interoperability, thanks to P4's flexibility in terms of protocol independence. This is one of the next steps of our activity.

Acknowledgements

This work was funded by the H2020 AI@EDGE project (https://aiatedge.eu; contract nb: 101015922) and the French ANR HEIDIS project (https://heidis.roc.cnam.fr; contract nb: ANR-21-CE25-0019).

References

[1] P. Bosshart et al. “P4: Programming protocol-independent packet processors”. In: *ACM SIGCOMM CCR* 44.3 (2014), pp. 87–95.

[2] *P4 Open Source Programming Language*. URL: https://p4.org/.

[3] *NetFPGA*. URL: https://netfpga.org/.

[4] S. Ibanez et al. “The p4-netfpga workflow for line-rate packet processing”. In: *Proc. of the 2019 ACM/SIGDA Intern. Symposium on FPGAs*. 2019, pp. 1–9.

[5] *P4-NetFPGA*. URL: https://github.com/NetFPGA/P4-NetFPGA-public/wiki.

[6] Xilinx. *P4-SDNet User Guide*. 2018. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1252-p4-sdnet.pdf.

[7] F. Hauser et al. “A survey on data plane programming with p4: Fundamentals, advances, and applied research”. In: *arXiv:2101.10632* (2021).

[8] D. Ding et al. “An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection”. In: *IEEE TNSM* 17.1 (2020), pp. 75–88.

[9] Y. Lin et al. “SDN Soft Computing Application for Detecting Heavy Hitters”. In: *IEEE TII* 15.10 (2019), pp. 5690–5699.

[10] V. Sivaraman et al. “Heavy-hitter detection entirely in the data plane”. In: *Proc. of the Symposium on SDN Research*. 2017, pp. 164–176.

[11] D. Ding et al. “In-Network Volumetric DDoS Victim Identification Using Programmable Commodity Switches”. In: *IEEE TNSM* 18.2 (2021), pp. 1191–1202.

[12] Z. Zhao et al. “HashFlow for better flow record collection”. In: *2019 IEEE 39th ICDCS*. IEEE. 2019, pp. 1416–1425.

[13] R. Ben Basat et al. “PINT: Probabilistic in-band network telemetry”. In: *Proc. of the Annu. Conf. of the ACM Special Interest Group on Data Comm. on the appl. technol. archit. and protocols for computer comm.* 2020, pp. 662–680.

[14] J. Marques et al. “Intsight: Diagnosing SLO violations with in-band network telemetry”. In: *Proc. of CoNEXT 2020*. 2020, pp. 421–434.

[15] D. Ding et al. “Estimating logarithmic and exponential functions to track network traffic entropy in P4”. In: *NOMS 2020-2020 IEEE/IFIP*. IEEE. 2020, pp. 1–9.

[16] P4.org API Working Group. *P4Runtime Specification, version 1.3.0*. 2021. URL: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html.

[17] R. Enns et al. “Network configuration protocol (NETCONF)”. In: (2011).

[18] N. McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM CCR* 38.2 (2008), pp. 69–74.

[19] *Protocol Buffers*. URL: https://developers.google.com/protocol-buffers.

[20] *gRPC - A High-Performance, Open-Source Universal RPC Framework*. URL: https://grpc.io/.

[21] S. Hollenbeck et al. “Guidelines for the use of extensible markup language (XML) within IETF Protocols”. In: *RFC3470* (2003).

[22] *TCAM (Ternary Content-Addressable Memory) on Verilog*. URL: https://github.com/mcjtag/tcam.

[23] *TCP Monitor*. URL: https://github.com/NetFPGA/P4-NetFPGA-live/tree/master/contrib-projects/sume-sdnet-switch/projects/tcp_monitor.

[24] *NetFPGA-SBI*. URL: https://github.com/cedric-cnam/NetFPGA-SBI.