MACSAD: High Performance Dataplane Applications On the Move

P Gyanesh Patra
University of Campinas, Brazil
gyanesh@dca.fee.unicamp.br

Christian Esteve Rothenberg
University of Campinas, Brazil
chesteve@dca.fee.unicamp.br

Gergely Pongracz
Ericsson Research, Hungary
gergely.pongracz@ericsson.com

***Abstract*—Deep programmability of dataplane pipelines is one of the tenets of the evolving Software Defined Networking (SDN) paradigm. Despite recent efforts on high performance programmable devices, achieving fully programmability (protocol independent) of heterogeneous dataplane implementations still pose numerous challenges. The P4 language is emerging as a strong candidate top-down approach to describe a protocol independent datapath pipeline, agnostic to network platforms. Meanwhile, the OpenDataPlane (ODP) project follows an open-source, bottom-up approach seeking multi-architecture APIs to write platform independent dataplane applications. In this paper, we present Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) as an approach to converge P4 and ODP through a common compilation process delivering portability of dataplane applications without compromising target performance improvements. We validate our prototype implementation through experimental evaluation of L2 and L3 dataplane applications on different target platforms (x86, x86+DPDK, ARM-SoC).**

I. Introduction

Traditional networking dataplanes are built around fixed packet processing pipelines yielding efficient designs in lieu of reconfigurability and flexibility. Support for new protocols commonly materializes through multi-year development cycles, often requiring new chip designs. This approach has at least two major drawbacks. Firstly, it is difficult for hardware manufacturer to fall back in case when a new protocol is not adopted. Secondly, researchers lack a common platform to design, develop, and test new algorithms and protocols to realistically evaluate innovative networking ideas.

Software Defined Networking (SDN) [1] changes the rigid layered dataplane pipeline design approach in support of flexible (match + action) abstractions for packet forwarding engines. This datapath generalization pushes the industry towards modern programmable chips based on relaxed table definition where size, number and functionality of tables become programmable. Early SDN protocols like OpenFlow were front-runners in capitulating the match plus action abstraction for device programmability but suffer from practical limitations due to the programmable knobs being largely dictated by the fixed functionality of the device pipelines (e.g. ASICs). As usual in embedded system designs, an increase in programmability through higher level of abstraction (i.e. generality) for chip (re)configuration adds complexity and overheads to the system design resulting in performance trade-offs.

978-1-5090-2839-9/17/$31.00

In theory, any programmable datapath can be defined by configuring the underlying tables in a specific way. Protocol Independent Switch Architecture (PISA) [2] is one such chip design approach that allows custom definition of the supported network protocols through programmable parsing and pipeline control. Approaching the challenge top-down through a Domain Specific Language (DSL) like Programming Protocol-Independent Packet Processors (P4) [3] based on high-level abstractions allow target-agnostic dataplane programmability. However, the backend compilation process is still less understood, and, as currently pursued, pushes the complexity down the stack to be solved on a per-target basis.

OpenDataPlane (ODP) [4] is a recent industry effort to provide an abstract API specification for dataplane applications. The APIs are vendor and platform neutral and span common features across different targets, enabling developers to write portable dataplane code.

By bringing P4 and ODP together, we can define and program dataplanes across multiple targets through a common compiler system that brings uniform development efforts for every platform. To this end, we propose MACSAD[1] [5] and allow developers to program P4-defined dataplanes seamlessly across multiple targets (i.e., dataplane applications "on the move"). In MACSAD, P4 is translated into high-level ODP APIs to deliver platform abstraction without compromising performance and hardware-acceleration options. The main contributions from this paper are the following:

- We present the MACSAD multi-architecture compiler system as a highly modular design capable of supporting new DSL and network platforms, among other features.
- We design a source to source compiler module to generate an Intermediate Representation (IR) for P4 applications.
- We implement a prototype and validate the *portability* of two P4 use case applications by automatically generating the datapath code i.e. "Datapath Logic" for heterogeneous targets (x86, x86+DPDK, ARM-SoC).
- We evaluate the *performance* of the different dataplane instances and applications in a 10G setup.

This paper introduces background technologies and related work in Section II followed by a detailed explanation of MACSAD architecture in Section III. Section IV presents the

[1]It is pronounced as 'Maksad' which means *purpose* or *motive* in Hindi. It is also simply referred as "MAC".

experimental use cases and evaluates the obtained results in terms of portability and performance. Limitations and future work are discussed in Section V prior to the conclusions and final remarks in Section VI.

II. Background and Related Work

A. Protocol Independent Switch Architecture

Bottom-up from the datapath perspective, PISA [2] chips are becoming a prominent programmable hardware approach. PISA architecture for chip design advocates that a programmable datapath can be defined by configuring the underlying tables, and, in turn, supporting (re)configuration of datapath/chip at a far later stage unlike during the fabrication phase as is the case of traditional networking ASICs while keeping the datapath simple and not compromising performance.

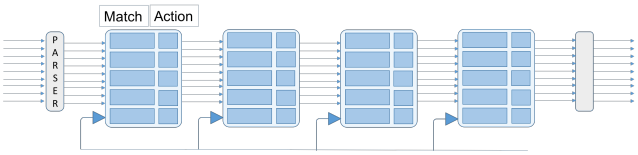


Fig. 1. PISA Architecture. Source: [2]

B. Programming Protocol-Independent Packet Processors

P4 is a declarative language which makes use of high-level network abstractions to allow dataplane programmability agnostic to the hardware target. By defining abstractions such as header, table, action, etc., P4 reduces the complexity of Dataplane Application (DAPP) programming turning the code base easier to write, understand, maintain, and debug. The abstract forwarding model shown in Fig. 2 illustrate how P4 allows to express a packet processing pipeline by programming the parser, match+action tables, and deparser functional blocks. When a packet arrives, the headers are parsed and then passed through the multi-table match and action pipeline before the deparser writes the headers back and sends the packet out. All these activities are defined inside the ingress and egress control flows defined by P4. Being protocol independent, it allows to define custom headers expressing arbitrary network protocol headers and fields. Table lookup methods over arbitrary fields can be defined along the actions to be applied upon a match.

C. Open Data Plane

The ODP [4] project aims at providing an abstract API specification to support Linux based network applications. ODP defines a set of high-level, common APIs spanning common features across multiple targets (see Table I) making dataplane applications portable. ODP can be compared to OpenGL as being the common standard for programming networking devices instead of video graphics. It can be considered as a higher abstraction than Data Plane Development Kit (DPDK) [7] and Netmap [8] extending to highly-optimized vendor-specific Software Development Kits (SDKs) while abstracting the hardware acceleration features (e.g., Crypto)

Fig. 2. P4 Abstract Forwarding Model. Source: Adapted from [6]

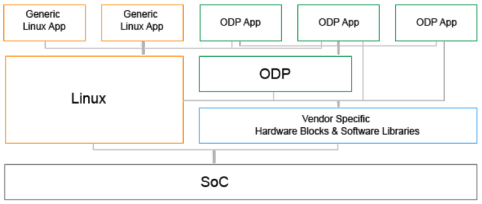


Fig. 3. ODP software stack in a Linux-based target. Source: [4]

of the underlying hardware. ODP can still use user-space fast packet processing I/O frameworks provided by DPDK and Netmap for improved performance. ODP thrives on the balance of SDKs to be open sourced vs left up to the semiconductor vendor. Figure 3 shows the scope of ODP in a switch platform complementing the vendor specific SDKs by providing common Application Programming Interfaces (APIs) turning ODP portable across platforms.

D. Related Work

Table II summarizes the main related projects around programmable switches. RouteBricks [9], GSwitch and CuckooSwitch [10] are some of the switches designed uniquely to provide higher performance using GPUs or different hashing methods (e.g. Cuckoo Hash) but lack a DSL approach. Microsoft's SONiC[2] is an open switch OS which supports various switch platforms by means of Switch Abstraction Interface (SAI) APIs but without a proper DSL. In contrast, OpenSwitch (OPENSWITCH)[3] and Netronome Network Flow Processor (NFP)[4] already offer platform-limited P4 support.

The PISCES [6] software switch is based on OpenvSwitch (OVS) [11] and allows the packet processing pipeline to be defined in P4. However, due to OVS pipeline constructs, there are restrictions around P4 and the dataplane reconfigurability.

Laki et al. [12] present a P4-based solution based on a Hardware Abstraction Layer (HAL) incorporating P4 to DPDK mapping. MACSAD follows a similar approach and

[2]https://azure.github.io/SONiC/
[3]http://www.openswitch.net
[4]http://open-nfp.org

TABLE I
ODP SUPPORTED PLATFORMS

Company	Supported Platforms
Cavium	ThunderX 24-48 core ARMv8 OCTEON TX 1-24 core ARMv8
Kalray	MPPA
Freescale	QorIQ ARM & PowerPC
Texas Instruments	Keystone2 Cortex A15
Linaro	PCIe NIC (odp-dpdk)

TABLE II
SCOPE, APPROACH, AND FEATURE COMPARISON LIST OF DIFFERENT PROGRAMMABLE SWITCH PROJECTS

Project	Protocol Independent	Development Effort	DSL Support	Target	Remarks
Click	Yes	Medium	No	General-Purpose Processor/ Server	Mostly used for research.
OVS	Limited	High	No	Software Switch	Can run as a part of Linux kernel.
Switchblade	No	High	No	FPGA	Verilog frontend
Cuckoo switch	Low	High	No	General-Purpose Processor/ Server	Cuckoo hashing used for FIB lookup
Routeshader	No	High	No	General-Purpose Processor/ Server	GPU-assisted packet processing. Good for IPSEC.
Routebricks	No	High	No	General-Purpose Processor/ Server	Use multiple CPU for packet processing.
Pisces	Yes	Low	Limited	Software Switch	OVS Based
P4 Switch*	Yes	Low	Yes	Limited by DPDK**	*http://p4.elte.hu **Optimized for Intel.
MacS	Yes	Low	Yes	Multi-Target	Currently X86 & ARMv8 support available.

uses the IR auto-generation methods as the starting point for "Transpiler". Our efforts fundamentally differ on the strong dependency on DPDK and the limited portability of DAPPS beyond general purpose processors, a strong feature of our ODP approach towards seamless portability.

III. MACSAD

The Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) is envisioned to achieve seamless portable DAPPS written in DSL (P4 being our starting focus) across network platforms while transparently leveraging hardware acceleration capabilities. While P4 gives means to the dataplane functionality, it is not responsible for the compiler system underneath. Manufacturers are compelled to add P4 support over their target-specific compilers. To achieve our primary goal of DAPP portability, there is a need for a common compiler system which understands different network platforms and allows to generate optimal code across targets. To this end, we propose to blend P4 and ODP by complementing their capabilities for a cross-platform compiler system.

The following design objectives are guiding our efforts on MACSAD development.

1) Fast and easy development environment of dataplane applications.
2) Dynamic & flexible pipeline by compiling a protocol independent DSL onto a hardware.
3) Bringing DAPP to different network platforms by adopting a common hardware abstraction layer without compromising with performance.

A. High-level Architecture

By mapping P4 network abstracts onto ODP APIs, which are high-level enough to allow platform abstraction without imposing strict models and overheads, we overcome the hazards of developing and maintaining target-specific compiler systems while maintaining performance and hardware-acceleration opportunities. To support different targets optimally, MACSAD(4a) architecture is designed around the following three modules:

Auxiliary Frontend: A plug-in framework to support different frontend DSLs, P4 being the initial choice focus.

Auxiliary Backend: Binds target-specific SDKs in order to support different platforms with ODP being the premier choice because of its openness and cross-platform nature.

TABLE III
PACKET PROCESSING FUNCTIONS

	Target Independent	Target Dependent
Primitive Actions(P4)	Add_header, copy_header, generate_digest, modify_field	Push, pop, count, meter
Pipeline Actions	Table Configuration, Protocol Independent Header Parsing	Pkt Rx/Tx, Header Parsing Modify Header Field, Table Creation, Table Lookup

Core Compiler: Composed of a *Transpiler* and a *Compiler* submodule, transforms the IR generated by the frontend into the target imaged in association with the auxiliary backend.

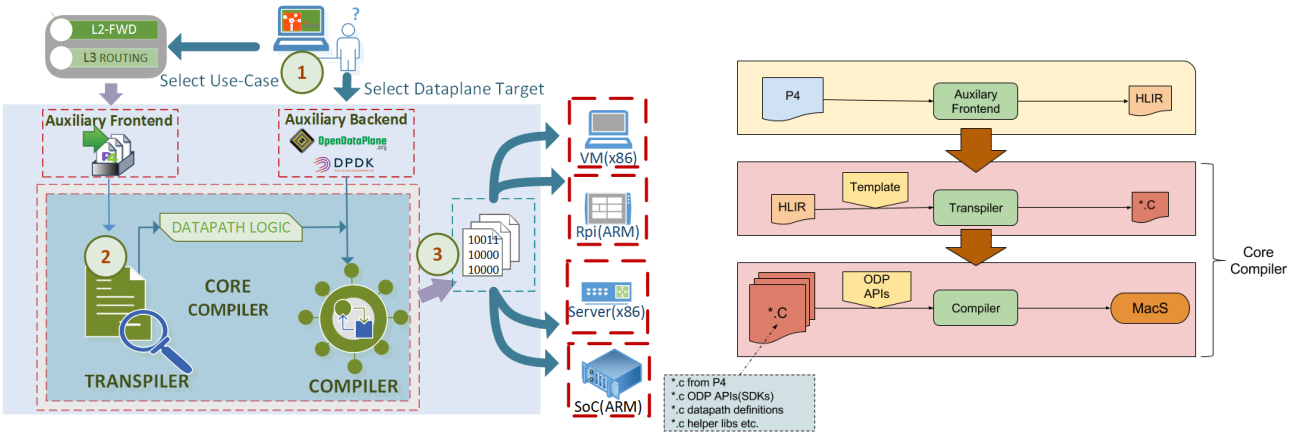
B. Separation of Concerns

P4 provides a list of primitive actions for handling network packets which are necessary to be mapped to ODP APIs in an effort for blending P4 with ODP. Apart from P4 actions, several other functions are necessary to manage packets in the data pipeline of a forwarding plane which need to be implemented in accordance with ODP APIs in MACSAD. To achieve portability and near optimal performance, we divide these functions into *Target Dependent* & *Target Independent* categories. Target Dependent functions can provide better performance when implemented optimally for a specific target, e.g., on a Linux system socket-mmap, dpdk & netmap are different options for packet I/O with varying performance. Table III shows a list of specific functions divided into these two groups. This division is not straight forward and requires careful considerations regarding different targets and use cases.

C. Auxiliary Frontend

The Auxiliary Frontend is designed as a plug-in framework allowing to add support for various DSL, with P4 leading the main developments. Accepting P4 program as an input, the Auxiliary Frontend creates an IR suitable for the 'Core Compiler' module complementing the official P4 frontend, i.e. `p4-hlir` project by P4 organization[5] which translates P4 programs into High-Level Intermediate Representation (HLIR). The top rectangle in Fig. 4b shows the generation of IR from DSL.

[5]www.p4.org



(a) Architecture & Use Case Workflow

(b) 3-Tier Compilation Process

Fig. 4. MACSAD Architecture

D. Core Compiler

The Core Compiler is the heart of MACSAD and comprises the *Transpiler* and *Compiler* sub-modules. It takes the IR generated by Auxiliary Frontend as an input and compiles it to MACSAD Switch (MACS)[6] for the target in association with the Auxiliary Backend.

1) Transpiler: Is a source-to-source compiler that takes input from the Auxilary Frontend and auto-generates the Datapath Logic codes in the final step of compilation process. The Datapath Logic is defined in 'C' language which is the 2nd level of IR of MACSAD. Datapath Logic is the formal definition of the pipeline in 'C' needed by 'Compiler' submodule. The middle rectangle in Fig. 4b demonstrates the auto-generation of 2nd level IR from HLIR using a template. The features of the Transpiler can be summarized as follows:

1) Responsible for transforming the loosely-typed DSL to a strongly-typed declaration while intelligently deciding on the data types depending on the underlying platform.
2) Creates a dependency graph of parser logic, tables and control flow enabling some code optimization like 'Dead Code Elimination' by identifying reachability in graph.
3) Decides the type of look-up mechanism to be used, and size & type of tables to be created by considering the resources available on the target platform.

2) Compiler: Generates the MACS for the target platform with ODP APIs and Datapath Logic IR code as input. It brings the regular array of optimization tools supported by the underlying compiler benefiting the consumer. We provide the support of GNU Compiler Collection (GCC) and Low Level Virtual Machine (LLVM) based compiler bringing a huge list of target support and optimization tools. Compiler sub-module also plays a big role in providing portability and improving performance as explained in Section IV.

E. Auxiliary Backend

Emerging programmable network platforms are breaking the fixed path ASIC paradigm, but are restricted by the lack of a unified compiler system. MACSAD fills this gap with the Auxiliary Backend module providing a common SDK for the Compiler incorporating the ODP APIs [4]. Newer platforms can be supported by implementing the ODP APIs over the new platform as ODP provides only the API specifications. Necessary libraries are developed as part of Auxiliary Backend to bridge the differences between P4 and ODP abstractions, and to support packet processing in dataplane pipeline. Target Dependent APIs can be implemented using the Target SDKs internally to leverage the hardware acceleration and other optimization features. This allows a developer to write applications using hardware acceleration features (such as Crypto) while being unaware of the nuances of the platform and its SDKs. The libraries of Auxiliary Backend span across packet I/O, resource handling, external controller support etc. This module also provides a placeholder for API auto-generation supporting various control protocols like SAI[7], OpenFlow (OF) [13], etc.

F. Portability

Portability of DAPP is a desired requirement being discussed for long time but so far unrealistically achievable in the wild. Attaining Portability is limited at least by two factors:

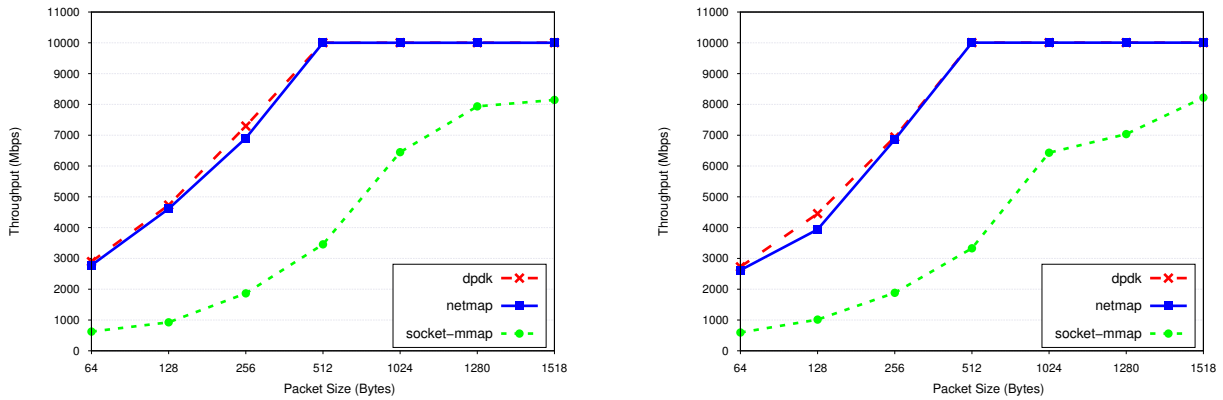
1) Lack of consent over a common programming language and SDK to configure different network platforms.
2) Absence of a common multi-architecture compiler system supporting different platforms.

P4 being an open adopter of PISA (but not limited to) advocates defining new protocols and datapath with higher network programming abstractions and achieves language and architecture separation. Hence the choice of P4 as the main programming language for our use cases. MACSAD is our answer to the second issue of absence of a common compiler system supporting a wide range of switching platforms. DAPPS written on P4 will be portable across platforms while exploiting hardware acceleration features.

Portability of DAPP can be achieved for almost every DAPP with different levels of effort needed to port at a certain

[6]The MACSAD compiled image is referred to as MACS.

[7]https://github.com/opencomputeproject/SAI



(a) L2-FWD (b) L3-Routing

Fig. 5. MACS measures throughput for different packet sizes and datapath use case applications.

amount of cost in terms of performance due to the port. With MACSAD, DAPPS should be portable seamlessly or with minimal effort and minimal impact over performance. This source portability is achieved by leveraging ODP APIs in the Transpiler and Auxiliary Backend modules.

IV. EXPERIMENTAL EVALUATION

In order to validate our MACSAD prototype implementation and the portability and performance claims, we carry two use cases experiments over different target platforms (x86, x86+DPDK, ARM-SoC) with just a recompile of source.

A. Testbed

The two use cases are evaluated using a two-device topology [14] where MACSAD compiled datapath DUT is connected to the Network Function Performance Analyzer (NFPA) [15] (aka. Tester) via two 10G links, from the sending port of the Tester to the receiving port of the DUT, and from the DUT sending port back to the Tester.

The DUT and Tester have a similar configuration (Intel(R) Xeon(R) CPU E5-2620 v2, 6 Cores, Hyper Threading Disabled, running at 2.1GHz, 8*8GB DDR3). Both the devices have a 1 Gbps management interface and a dual 10Gbps NIC (Ethernet Controller 10-Gigabit X540-AT2). To generate test traffic, the NFPA test system internally uses PktGen [16] tool with DPDK whereas MACS is tested with socket-mmap, Netmap and DPDK packet I/O to illustrate the ability to accommodate various different platform features.

B. Use Case Applications

1) L2-FWD: A Layer-2 forwarding switch program in P4 where an external controller performs MAC learning and populates the L2 tables. MACSAD creates the MACS which learns new MAC address and corresponding port bindings by generating and sending a digest to the controller which in turn updates the corresponding source and destination MAC address lookup tables. The P4 "Exact Lookup" method is implemented with a Cuckoo Hash algorithm.

2) L3-FWD: Layer-3 routing with Longest Prefix Match (LPM) based on IP address lookup mechanism. MACS does a destination IP address lookup for the incoming traffic and forwards them to the proper destination, updating the TTL and the source and destination MAC addresses.

C. Performance Analysis

Figures 5a and 5b present the measured throughput (Mbps) for different packet sizes. We can observe that Linux Socket-mmap performed 4 to 5 times slower compared to Netmap and DPDK, which reached 10G line speed with packet sizes of 512 Bytes and above. We can observe that the performance behavior for both use cases are similar and can be explained by the effectiveness of the system memory caches under a simple workload (single host destination).

Modern NICs support multiple hardware queues to avoid resource contention. Multi-core programming benefits this by defining affinity among CPU cores to queues to improve performance by reducing average CPU stall time. In our configuration of n NICs, each core is mapped to 1 Rx and $(n-1)$ Tx queues. Hence a core can receive from only one NIC it is mapped to and it can transmit to every other NIC.

In order to understand the current system limits around smaller packets where line rate is not achieved, we measure the forwarding rate of 256-Byte packets with an increasing number of cores (see Fig. 6). In our setup, we observe a performance peak ($\approx 30\%$) when the number of physical cores are around the number of NICs. Increasing the number of cores leads to higher cache invalidation and forced main memory accesses. Higher Rx queue counts often result in empty queue or fewer packets than burst size while polling which makes thread context switching costlier than amortization of packet processing cost. Also with an increase in Tx, queue counts NIC controller requires better packet scheduling to transmit. These factors contribute to explanation of the performance decrease with more cores, a design objective we will further work on.

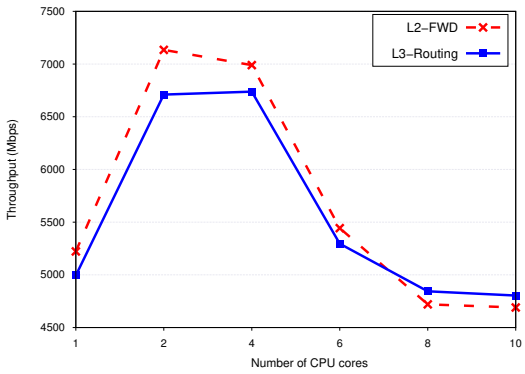


Fig. 6. Throughput with increasing number of cores.

D. Discussion and lessons learned

The first noteworthy remark on the obtained results is that ODP does not provide any inherent table management/lookup mechanism, contrary to DPDK which has various fully optimized table management libraries. Hence the experiments are carried with arguably suboptimal user-space table implementation, which may have an impact on the observed performance. Furthermore, the current DPDK packet I/O implementation in ODP does a packet copy between ODP buffer and DPDK hitting the attainable throughput.[8]

We now discuss a number of methods contributing to performance. To begin with, the adoption of *inline functions* in the auto-generated datapath logic code has a positive impact on the datapath speed. As forwarding datapath consists of a set of functions invoked for every packet, inlining these functions brings notable performance gains. Secondly, we opt to transmit *packets in batches* at outgoing interfaces. Initially, each packet was handled to completion before processing the next packet incurring into expensive memory-mapped I/O (MMIO). By batching packets and sending them out periodically (or when a vector threshold is reached, e.g., batch size equals 32) introduces further performance improvement by amortizing the cost of transmitting a single packet.

V. Limitations and Future Work

As today, MACSAD supports DAPPS written in the current stable P4 version –a partial set of P4 abstractions are supported and more are being added continuously, mainly driven by selected use cases. The next P4 version ($P4_{16}$) is around the corner with a publicly released draft. $P4_{16}$ brings language-architecture separation and should be able to program not only switches but also firewalls, NIC, VNFs and so on. $P4_{16}$ removes the architecture related P4-programmable blocks (e.g. ingress/egress pipeline, deparser, state full memories etc.) from the core language and provide them as an architecture library part of an architecture model defined on a per network platform. It also supports extern types, libraries (code reuse) and strong data types.

Towards further DAPPS portability, efforts are being devoted to the Cavium ThunderX platform and its various hardware acceleration features. Furthermore, we want to improve the auto-code generation features and integrate auto-optimization techniques for the generated code. Adding binary portability support is also in our agenda, along new and real-world use cases with realistic (and worst-case) workloads to assess the maturity and applicability of MACSAD.

[8]We are currently working on an upgrade to integrate the PCIe NIC optimized implementation `odp-dpdk` (https://git.linaro.org/lng/odp-dpdk.git) which will help to remove the packet copy cost

VI. Conclusions

MACSAD explores a novel approach towards dataplane programmability by striking a niche balance between portability and performance. The proposed Transpiler module is capable of auto-generating Datapath Logic from P4-defined programs following a language-architecture split approach conserving platform-specific optimizations through Auxiliary Backends. The separation of target-dependent from independent dataplane functions allows, when available, to leverage hardware acceleration features of the target platforms. The L2-Fwd and L3-Routing use cases presented in this paper serve as promising evidences towards dataplane portability, i.e., the same DAPPS seamlessly running over various network platforms at current edge technology performance.

References

[1] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, 2015.
[2] *Protocol-independent switch architecture*. [Online]. Available: http://schd.ws/hosted_files/p4workshop2015/c9/NickM-P4-Workshop-June-04-2015.pdf
[3] P. Bosshart et al, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, Jul. 2014.
[4] Opendataplane. [Online]. Available: http://www.opendataplane.org
[5] P. G. Patra, C. E. Rothenberg, and G. Pongrácz, "MACSAD: Multi-Architecture Compiler System for Abstract Dataplanes (Aka Partnering P4 with ODP)," in *ACM SIGCOMM'16 Demo and Poster Session*, 2016.
[6] M. Shahbaz et al, "PISCES: A Programmable, Protocol-Independent Software Switch," in *ACM SIGCOMM*, 2016.
[7] Intel. dpdk: Data plane development kit. [Online]. Available: http://dpdk.org
[8] L. Rizzo, "netmap: A novel framework for fast packet i/o," in *USENIX ATC 12*.
[9] M. Dobrescu et al, "Routebricks: exploiting parallelism to scale software routers," in *ACM SIGOPS*, 2009.
[10] D. Zhou et al, "Scalable, high performance ethernet forwarding with cuckooswitch," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13, 2013.
[11] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *NSDI*, 2015.
[12] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, and M. Tejfel, "High speed packet forwarding compiled from protocol independent data plane specifications," in *ACM SIGCOMM'16 Demo and Poster Session*, 2016.
[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, 2008.
[14] S. Bradner, "Benchmarking methodology for network interconnect devices," RFC 2544, March 1999.
[15] L. Csikor et al, "NFPA: Network function performance analyzer," in *IEEE NFV-SDN*, 2015.
[16] D. Turull, P. Sjdin, and R. Olsson, "Pktgen: Measuring performance on high speed networks," *Computer Communications*, 2016.