Towards a Sweet Spot of Dataplane Programmability, Portability and Performance: On the Scalability of Multi-Architecture P4 Pipelines

P Gyanesh Kumar Patra, Fabricio E Rodriguez Cesen, Juan Sebastian Mejia, Daniel Lazkani Feferman, *University of Campinas, Brazil,* Levente Csikor *Budapest University of Technology and Economics,* Christian Esteve Rothenberg, *University of Campinas, Brazil,* and Gergely Pongracz, *Ericsson Research, Hungary*

***Abstract*—Despite having received less attention compared to the control and application plane aspects of Software-Defined Networking (SDN), the data plane is a critical piece of the puzzle. P4 takes SDN datapaths to the next level by unlocking deep programmability through a target-independent high-level programming language that can be compiled to run on a variety of targets (e.g., ASIC, FPGA, GPU). This article presents the design and evaluation of our sweet spot approach on SDN datapaths offering three contending characteristics, namely, *performance, portability* and *scalability* in multiple realistic scenarios. The focus is on our Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) proposal, which blends the high-level protocol-independent programmability of P4 with low-level but cross-platform (HW & SW) APIs brought by OpenDataPlane (ODP), this way supporting many different vendors and architectures. Besides the performance evaluation for varying packet sizes and memory lookup tables, we investigate the impact of increasing pipeline complexity ranging from elemental L2 switching to more complex data center and border network gateways. We investigate the scalability for increasing number of cores and evaluate a novel method for run-time core reallocation. Furthermore, we run experiments on different target platforms (e.g., x86, ARM, 10G/100G), inducing different ways of packet mangling through specific drivers (e.g., DPDK, Netmap), and compare the results to state-of-the-art datapath alternatives.**

***Index Terms*—SDN, ODP, P4, Programmable Networks, IPv6.**

I. Introduction

Software-Defined Networking (SDN) [1] tries to break the dogma of networking equipments by proposing a clear and programmatic separation between control and data plane functions. The OpenFlow protocol [2] has been fundamental to unlock traditional thinking on control plane split aiming at a *de facto* standard for programmable data plane devices. After one decade since its inception, the current state of affairs points to a less promising future, mainly attributed to hardware compatibility and evolvability issues, fragmented optional features support among vendor implementations, etc. [3], [4].

The benefits and arguably success of SDN are commonly attributed to control plane innovations, relegating datapath components to a deuteragonist role. Lessons learned during the path towards rich data plane programmability at the crossroads of new switch architecture designs result in new promising ways to take a software compiler approach to the functional definition of datapath pipelines and their APIs to the control/management plane. The main ongoing trends towards deep programmable data planes can be synthesized as follows:

· Programmable hardware designs supporting custom protocol stacks with methods such as relaxed table definitions and *match + action* abstractions as pursued by so-called Protocol Independent Switch Architecture (PISA) designs.

· Top-down approaches in spirit of a Domain Specific Language (DSL) to provide high level abstractions to describe forwarding policies and datapath pipelines with Programming Protocol-Independent Packet Processors (P4) [5] being the main example of target-agnostic data plane programmability.

· Bottom-up efforts on platform-agnostic low-level Software Development Kits (SDKs) for datapath chips to foster portability. One relevant approach in this field is OpenDataPlane (ODP) [6], an open source project on vendor-neutral abstract APIs covering common features across several targets, however its low-level programming interface cannot keep up with today's agile prototyping and programming needs.

Considering the landscape, our work aims at blending P4 with ODP to create a compiler system, called MACSAD,[1] to provide a *high-level and cross-platform* portable data plane application compiler in response to limited availability of open source, protocol-independent and programmable data plane solutions delivering high performance. After our initial proof of concepts at small scale [7], [8], our main efforts have been devoted to increasing the feature completeness of MACSAD regarding different versions of P4 ($P4_{14}$ and $P4_{16}$ support), the design of more complex use cases, evaluation of portability, performance and scalability, which are in the main focus and contributions of this article:

(*i*) We validate MACSAD design and prototype by compiling P4 applications into different target platforms (x86, x86+DPDK, ARM-SoC), resulting in effective *portability* of the auto-generated datapath code.

(*ii*) We implement several use case pipelines of increased complexity (Ethernet switching, IPv4/v6 Forwarding, VxLAN-based Data Center Gateway, and Broadband Network Gateway) leveraging $P4_{14}$ and $P4_{16}$ *programmability*.

(*iii*) We carry an extensive *performance* and *scalability* evaluation of all use cases for varying test workloads (packet traces, table sizes) and target platform configurations (e.g., I/O, 10/100G NICs, CPU type and #cores).

(*iv*) We investigate a novel technique for run-time *scaling up/down* the number of cores allocated to packet processing.

[1] It is pronounced as 'Maksad' which means *purpose* or *motive* in Hindi.

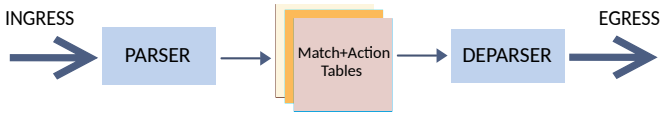


Fig. 1. P4 Abstract Forwarding Model of a Datapath Pipeline.

Background technologies are introduced in Sec. II followed by a detailed explanation of the MACSAD architecture in Sec.'III. Section IV discusses the experimental use cases and evaluates the obtained results in terms of portability and performance. Related works are summarized in Sec. V. Finally, conclusions and future work are discussed in Sec. VI.

II. Background

Protocol Independent Switch Architecture. The packet processing paradigm behind PISA designs[2] is based on programmable hardware-based datapaths by means of pipeline (re-)configuration of a chained set of *match+action* tables. Compared to traditional ASIC-based datapath designs, PISA delivers rich flexibility without compromising performance for comparable chip area and energy consumption.

Programming Protocol-Independent Packet Processors. P4 is a DSL to describe datapath packet processing pipelines using an abstract model (see Fig. 1) in spirit of PISA. P4 language constructs include high level networking abstractions remaining agnostic to the actual targets, which can be either based on hardware or software implementations. Based on the abstractions (such as header, table, action, etc.), a P4 pipeline consists of parser, *match+action* tables, and deparser functional blocks, in particular packet headers are parsed upon arrival, processed through a multi-table pipeline, and deparsed into a packet to be sent out.

OpenDataPlane. ODP [6] is an open source project aiming at a network data plane Application Programming Interface (API) specification for developers to design data plane applications. ODP defines a set of unified APIs covering common standard features across diverse target platforms including ARM (e.g., Cavium), Power PC (e.g., Freescale), and x86 (e.g., Linaro) allowing application portability. An application developed using the "ODP implementation" can leverage vendor-specific features of the underlying platform (e.g., hardware accelerations features) by means of the "Vendor Specific Hardware and Software Libraries". Developers can take advantage of the joint presence of vendor-optimized ODP and vendor-specific libraries to write data plane application using platform-specific features not part of an ODP implementation. The ODP Helper Library offers commonly used functions such as those related to hash table, IP lookup (IPv4-only), thread management, etc. In contrast, DPDK [9] offers multiple fully optimized table management libraries. Being a higher abstraction, ODP natively supports high-performance Linux user-space based packet I/Os such as DPDK and Netmap [10].

III. MACSAD

Our MACSAD proposal [7], [8] aims at hiding data plane programming complexity using P4 while keeping the flexible data plane portable and scalable through the performance and hardware acceleration features of ODP. From an implementation aspect, it merges protocol-independent P4 abstractions and primitives with ODP APIs towards data plane applications.

[2]http://schd.ws/hosted_files/p4workshop2015/c9/NickM-P4-Workshop-June-04-2015.pdf

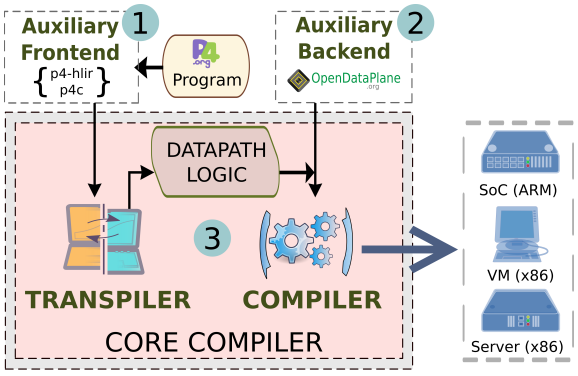


Fig. 2. High-level MACSAD Architecture. Adapted from: [7]

A. Architecture details

The high-level architecture of MACSAD in Fig. 2 is divided into modules in sought for *protocol independence* and *target independence*.

1) Auxiliary Frontend: The Auxiliary Frontend transforms a P4 program into an Intermediate Representation (IR) suitable for the 'Core Compiler' module by integrating projects from P4 consortium. It integrates `p4-hlir`[3] to translate $P4_{14}$ programs into High Level Intermediate Representation (HLIR) while, at the same time, it creates JSON IR for $P4_{16}$ programs using `p4c` compiler. The top left rectangle in Fig. 2 depicts the transformation of P4 program into IR and passed to the Transpiler submodule.

2) Auxiliary Backend: The Auxiliary Backend comprises of all internal and helper APIs turning MACSAD into a unifying compiler system addressing through a common SDK based on ODP APIs. It implements all necessary APIs using ODP APIs to support P4 abstractions. Adding support to a new target platform is equivalent to just porting the ODP APIs to the new platform.

Another aspect shown in Table I is to provide target-dependent APIs, leveraging hardware acceleration and optimization features, when available. Packet I/O, packet manipulation, resource handling, controller support etc., are some groups of helper and internal APIs of the Auxiliary Backend.

3) Core Compiler: The Core Compiler is the heart of MACSAD and encompasses the Transpiler and Compiler components. The IR received from the the Auxiliary Frontend is compiled with the ODP APIs provided by the Auxiliary Backend into MACSAD Switch (MACS) (hereafter, the MACSAD compiled binary code is referred to as MACS throughout the text) for the desired target platform.

a) Transpiler. In MACSAD the data plane code consists of two pieces of code where the first one is auto-generated and the second one is written as a part of the Auxiliary Backend submodule through helper and internal APIs. The Transpiler

[3]https://github.com/p4lang/p4-hlir

TABLE I
PACKET PROCESSING FUNCTIONS

Target-Independent	Target-Dependent
add_header, remove_header, copy_header, generate_digest, modify_field, Table Configuration, Protocol Independent Header Parsing	push, pop, count, meter, Pkt Rx/Tx, Checksum, Modify Header Field, Table Creation, Table Lookup

is our template based source-to-source compiler solution to output the auto-generated code written in 'C' consisting of the *Packet Parsing Logic* which includes data structures for 'header fields, their offset & bitmasks' and handles packet header parsing, and the *Control Logic* for the packet flow across the tables defined in the P4 program. The *Control Logic* implements the target-independent functions (see Table I). The main activities of the Transpiler during the source-to-source compilation consists of *(i)* defining table constructs (e.g., size, lookup); *(ii)* creating *Packet Parsing Logic* based on the IR from III-A1; *(iii)* mapping the loosely-typed DSL (i.e., P4) to strongly-typed (i.e., 'C') declarations[4] in auto-generated code by selecting appropriate data types considering the target platform; *(iv)* taking performance optimization decisions (e.g., RX burst size) based on predefined platform specificities.

b) Compiler. The Compiler submodule sits at the final stage of MACSAD and is responsible for the binary code generation of MACS using III-A2 and output of III-A3 with the underlying GNU Compiler Collection (GCC) / Low Level Virtual Machine (LLVM) compilers.

B. Advantages of the Architecture

1) Protocol Independence: Protocol independence is a forte that is achieved by being able to (re-)configure data plane using DSL (P4). The *Packet Parsing Logic* and the *Control Logic* auto-generated by the Transpiler are responsible to bring Protocol Independence (PI) to MACSAD by means of a *Protocol Independent Parser* and a *Protocol Independent Dataplane*, respectively.

P4 abstract model (see Fig. 1) can be interpreted as post-pipeline editing where the "Parsed Representation" of packet headers are updated and pushed back to the packet by deparser module. In contrast, MACSAD *Protocol Independent Parser* parses each packet and stores the pointers to the headers and header fields, and follows inline editing of headers circumventing the deparser module to improve performance.

Moreover, these auto-generated header structures are used by the *Control Logic* functions while avoiding any standard protocol header structure to achieve *Protocol Independent Dataplane*. These *Control Logic* functions include all target independent functions for table configuration, the *match+action* logic and header update functions as per Table I.

2) Portability: A datapath implementation in software typically consists of two functional realms: *(1)* Packet handling consisting of the Parser, Table (*Match+Action*) lookup, and Packet header updater; and *(2)* Switch resource management functions including CPU, Queue, Memory, Thread, Table, among others. The first set of functions are mostly auto-generated by the Transpiler in a protocol-independent manner. Though the second set is target-dependent, MACSAD target-independent implementation of it built upon libraries/APIs on top of ODP APIs, turn the resulting code seamlessly (or at highly-reduced effort) portable across network platforms.

[4]Due to providing high-level abstractions and fundamentally one type of variable, we considered P4 as a loosely-typed language

IV. PERFORMANCE AND SCALABILITY EVALUATION

We now turn the attention to the practical aspects of the MACSAD prototype implementation and evaluate the portability and performance for five different use cases with increasing complexity,[5] namely, Layer-2 forwarding (L2-FWD), Layer-3 forwarding with IPv4 (L3-IPv4) and IPv6 (L3-IPv6), Data Center Gateway (DCG), and Broadband Network Gateway (BNG) using three different target platforms (x86, x86+DPDK, ARM-SoC) and three different packet I/O engines (DPDK, Netmap, Socket_mmap).

The main aim of our experimental evaluation is to identify how our proposed MACSAD performs under the same circumstances compared to its pure P4- and ODP-based counterparts. We execute MACS along with an in-house simple controller to populate the tables for each use case. We present and analyze the results and discuss the observed trade-offs and scalability patterns for different workloads (packet traces, table entries) and configuration options (e.g., CPU cores). Finally, we evaluate a novel technique on dynamic CPU core allocation towards adaptive and scalable data planes.

For further implementation details and reproducibility purposes, MACSAD[6], all P4 use case programs,[7] and the packet and trace generator tool BB-Gen [11] are publicly available.[8]

A. Testbed and Methodology

Our testbed includes two servers with Intel Xeon E5-2620v2 processors (6 cores, HT-disabled), dual-port 10 G Intel X540-AT2 NIC and 64GB of memory running Ubuntu Linux 16.04 LTS (kernel 4.4). One server (Tester) runs Network Function Performance Analyzer (NFPA) [12] with a recent stable version of DPDK (v17.08) and PktGen (v3.4.5) connected back-to-back with the Device Under Test (DUT) [13]. The DUT supports multiple packet I/Os, namely DPDK (v17.08), ODP (v1.16.0.0.), Netmap (v11.2), and Linux Socket_mmap.

With the testbed configuration, packet loss only occurs when the DUT becomes a physical bottleneck and therefore the packet rate received by NFPA is representative of the raw performance. Traffic traces have different number (from 100 to 1M) of unique flows, randomly generated per use case experiment run but consistent across different packet sizes, limiting the impact of the lookup process and underlying caching system which would depend on the traffic pattern. In all cases, we evaluated different packet I/O drivers for which, when it is not stated otherwise, we used blue circle

[5]It means increasing table count (from a couple to 10s) and header fields to match on (from 100 to 1M), i.e., increasing per packet processing time

[6]https://github.com/intrig-unicamp/macsad

[7]https://github.com/intrig-unicamp/macsad-usecases

[8]https://github.com/intrig-unicamp/BB-Gen

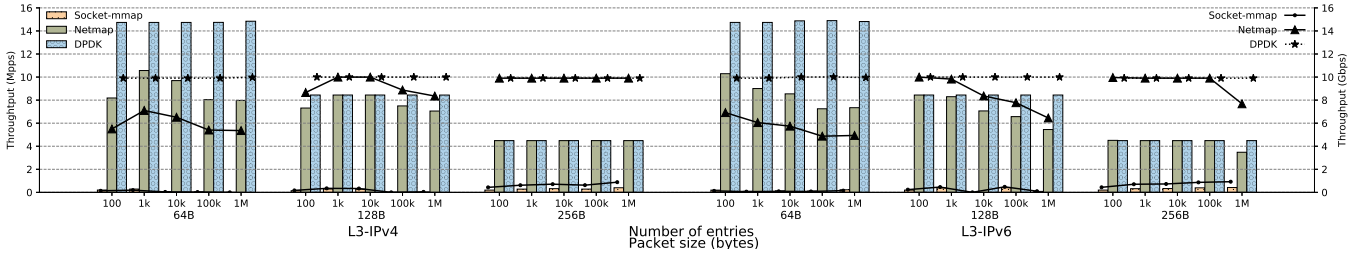


Fig. 3. IPv4 and IPv6 forwarding performance of different I/O drivers for different packet sizes and table entries (4 CPU cores).

patters for DPDK, green solid bars for Netmap, and orange dotted pattern for the kernel provided Socket_mmap. All measurements were conducted for 60 sec [13], and every data point in our performance measurements is an average value. Confidence intervals are unnecessary as results are stable and reproducible for all frameworks.

B. Layer-2 forwarding use case (Ethernet)

L2-FWD implements two separated lookup tables, the first matching on incoming port and source MAC and the second on destination MAC addresses. MACS generates controller digests for new MAC addresses and port binding to update the lookup tables. P4 "Exact Lookup" methods are implemented using ODP based Cuckoo Hash algorithm. For the L2-FWD, using DPDK with a packet size of 64 bytes and 4 CPU cores, we reached line rate (i.e., 14.9 Mpps) with flow table sizes of 100 and 1K entries and observed a performance decrease to about 7.2 Mpps for 10k entries. A similar behaviour was observed with Netmap and Socket-mmap.

C. Layer-3 forwarding use cases (IPv4/v6)

The IP routing use cases are implemented with ODP's built-in Helper library for Longest Prefix Match (LPM) based lookup mechanism with 32-bit and 128-bit keys supporting IPv4 (L3-IPv4) and IPv6 (L3-IPv6) forwarding. The P4 pipeline consists of two tables; IP lookup is performed in the first table along with corresponding actions of standard L3 packet processing (e.g., MAC_{dest} re-writing, TTL/Hop Limit decrement, Output Port selection) followed by a matching on the output port in the second table with the MAC_{source} re-writing action. For brevity, we focus on the results for 4 cores and leave the multi-core scalability discussion for Sec. IV-G.

1) L3-IPv4: The IPv4 LPM implementation in MACSAD uses a binary tree lookup process with three levels (16-8-8) to achieve balance between memory consumption and lookup speed (bounded at 3 memory accesses per lookup).

Fig. 3 shows the observed performance for different Forwarding Information Base (FIB) sizes and packet I/O drivers. On the two y axes, the achieved throughput is shown in Mpps (left) and Gbps (right). It can be observed that MACS with DPDK saturates the 10G interface even with the smallest packets (64 bytes) irrespective of the FIB table size. Lower yield for Netmap with 64B and 128B packets confirms to previous literature [14]. Notable, the measured results for 1K FIB entries are better than for 100. This can be caused by the suboptimal use of the CPU queues with small packet sizes (64 Bytes) and by the number of packets as observed in Fig. 3. As expected, the Linux Socket_mmap driver stands last and never saturates the 10G interfaces.

2) L3-IPv6: Since the original ODP Helper library did not support IPv6, we implemented the required LPM module similar to that of DPDK[9] with 15 levels of tables (16-bit 1^{st} level followed by 14 levels of 8-bit each). As shown in Fig. 3 (right), performance results are on par with the findings of L3-IPv4. While DPDK reaches line rate with 64B packets for any FIB size, Netmap performance drops with increasing FIB size (i.e. more table entries) due to higher TLB misses (DPDK keeps TLB misses under control by using Hugepages). Also noteworthy, the anomaly for 100 and 1K FIB entries observed for L3-IPv4 does not apply for L3-IPv6.

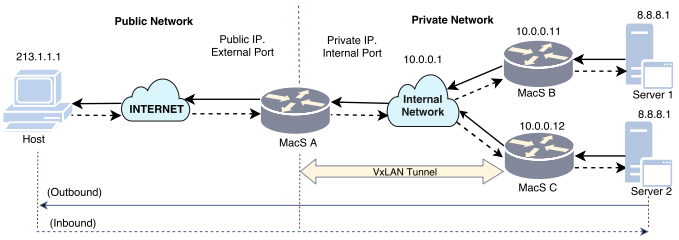


Fig. 4. Data Center Gateway (DCG) use case using VXLAN tunnels.

D. Data Center Gateway (DCG) with VXLAN

The DCG use case VXLAN tunnels are used to connect Internet hosts with (virtualized) Web services in redundant servers sharing a common IP address (8.8.8.1 in our example). The VXLAN protocol provides an encapsulation mechanism between Virtual Tunnel End Points (VTEPs) to transport L2 frames inside UDP packets. The experimental scenario shown in Fig. 4 can be divided into two sub-cases:

1) Inbound (IB): An Internet Host (213.1.1.1) sends traffic to a web service (8.8.8.1). When the originating packet reaches the data center gateway (MACS A VTEP), a load balancing next hop VTEP decision is taken (e.g., MACS B) and the actions carried to set the outer L2 (MAC_{dest} of B), L3 headers (IP_{dest} set to 10.0.0.11), UDP and VXLAN headers. In turn, as the last leg of the VXLAN tunnel, MACS B decapsulates the packet and sends it to Server 1.

[9]http://dpdk.org/doc/guides-16.04/prog_guide/lpm6_lib.html

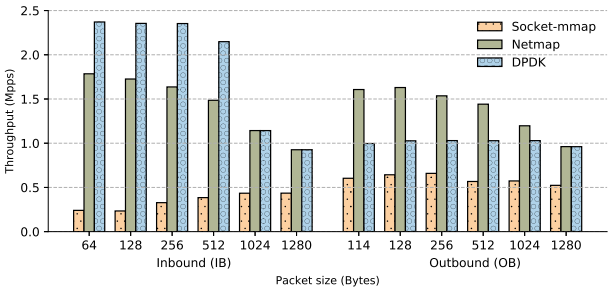


Fig. 5. Performance comparison of the DCG implementation for different I/O drivers (4 cores, 100 table entries).

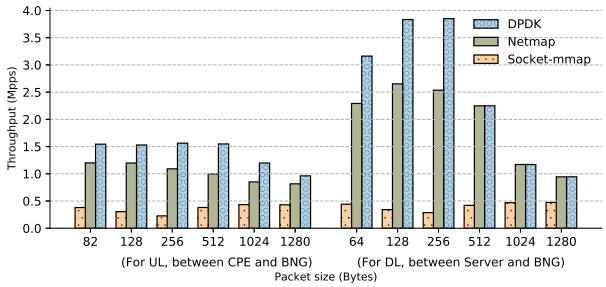


Fig. 6. Performance comparison of the BNG implementation for different I/O drivers (4 cores, 100 table entries).

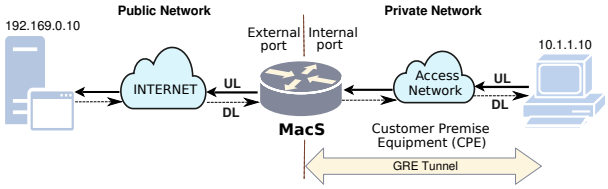


Fig. 7. BNG use case illustrating a subscriber and an external public service.

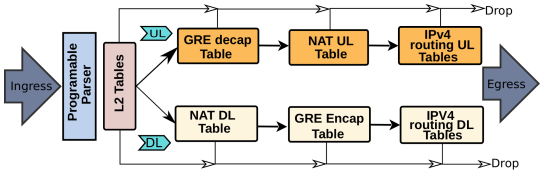


Fig. 8. P4-defined BNG pipeline featuring the main UL and DL tables.

2) Outbound (OB): As a response, Server 1 sends a response packet that MACS B VTEP encapsulates in the reverse direction towards MACS A VTEP, which removes the VXLAN header, rewrites addresses and forwards the packet.

Towards a sophisticated, scalable processing and to avoid the cross-product problem [15], the pipeline consists of multiple matching tables (8 for IB and 7 for OB). The learning switch table is pre-populated and the load balancing feature is implemented by a checksum function using source IP address. The VXLAN encapsulation adds the right headers and port numbers prior to MAC address re-writing. Altogether, IB and OB matches 7 and 6 tables respectively while remaining tables use the default action `_nop` to move to the next table.

The used traffic trace includes packets with random host IPs and a fixed server destination IP (set to 8.8.8.1). For brevity, we show results of IB and OB sub-cases in Fig. 5 as a function of increasing packet sizes, 100 FIB entries and with 4 cores. One can observe that for packet sizes greater than 1024B, the throughput attains the line rate (10G). Similar measurements are presented in Fig. 9 for 128B packets, 2, 4 and 6 cores. VXLAN encapsulation in the middle of the pipeline for IB sub-case refreshes cache which is leveraged by tables further down in the pipeline, where as decapsulation happens at the end of the pipeline for OB resulting in higher cache miss. This results in poorer DPDK performance compared to NETMAP as DPDK has a higher cache footprint [14].

*E. Broadband Network Gateway (*BNG*)*

BNG, also known as Broadband Remote Access Server (BRAS), is a quintessential part of today's Internet as it handles the majority of access network traffic implementing network policies and services that an Internet Service Provider (ISP) defines per subscriber. Functions of a BNG include: Authentication, Authorization and Accounting (AAA) and session management; Packet en/de-capsulation; ARP proxy; NAT; QoS enforcement. Fig. 7 illustrates a BNG use case handling traffic between a Private and an External (public) network, with the main data plane functions divided into an Upload (UL) and a Download (DL) pipeline.

This MACSAD use case is implemented with $P4_{16}$, which offers a more stable (future-proof) and simpler data plane definition compared to previous $P4_{14}$. Figure 8 illustrates the implemented pipeline using multiple sets of tables as follows:

L2/Ethernet. This set of tables allows BNG to act as a L2 learning switch and to processes ARP packets coming from the CPE. Additionally, it helps to set the network interface either to external or internal to separate the UL and DL traffic.

NAT UL/DL. Since the CPE and its users are residing within a private IP network, NAT is required to translate IPv4 address and TCP ports. Packets without corresponding entries in the NAT table are dropped.

GRE Encap/Decap. For DL traffic, the Generic Routing Encapsulation (GRE) Encap table encapsulates packets destined to the external network with a GRE packet header [16] identifying the user to establish an user session. In the reverse direction (UL), CPE-originating packets are decapsulated. The header *add* and *remove* methods are examples of functions implemented in the Auxiliary Backend using ODP APIs.

IPv4 UL/DL. Implemented as in the L3-IPv4 use case.

Two types of traffic traces were used: UL path coming from the Customer Premise Equipment (CPE) (IP address 10.1.1.10) to an Internet server (IP: 192.169.0.10), and DL path from the server back to the CPE.

1) UL: A home gateway encapsulates the CPE traffic with GRE towards the access network to MACS BNG. MACS performs L2 address learning, verifies user ID, and decapsulates the GRE packet. The NAT table rewrites inner headers with the appropriate source IP address and TCP ports towards an external destination server (192.169.0.10).

2) DL: The server (192.169.0.10) sends TCP traffic back to the user client (10.1.1.10) via MACS through the external interface. The MACS BNG performs NAT, adds the point-

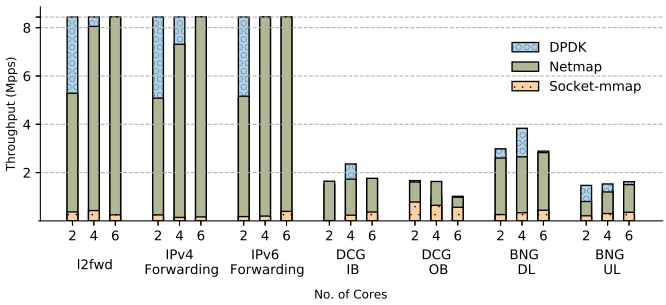


Fig. 9. Use case performance comparison (100 entries, 128 byte packets).

to-point GRE tunnel header, writes the IPv4 outer header, verifies the user ID[10] and completes the IP packet forwarding selecting the next hop and output port towards the home gateway performing GRE decapsulation.

The traffic traces include unique addresses for source Ethernet, inner source and outer destination IPv4, as well as TCP ports. All flow tables are pre-populated by the controller based on the test traces. Results with 100 packet flows and 4 cores are shown in Fig. 6 (left for UL and right for DL)[11]. Similarly measurements for 128B packets, 2, 4 and 6 cores are presented in Fig. 9. As expected, DPDK yields the best performance [14] for each packet size but due to the pipeline complexity (i.e., GRE, number of tables), MACSAD was only able to saturate the 10G link for packets larger than 1024B.

F. Scalability and Portability: From x86 to many-core ARM

Scalability. Fig. 9 shows a throughput comparison among the use cases with increasing complexity in terms of number of table lookups and packet actions. DCG and BNG throughput results confirm that with increasing number of table lookups and tunneling support, the performance takes a hit, i.e., table lookups are not free in terms of processor cycles and memory accesses. An interesting observation is that for the more complex use cases (DCG and BNG) the performance does not scale in par with the number of CPU cores as for the L2-FWD and L3-IPv4/v6 use cases. We observe that as the CPU takes more cycles to process packets, the system overload has a negative impact resulting in a throughput drop when the number of cores increases from 4 to 6.

For the simpler use cases, we examined the scalability on a different NUMA architecture as well while comparing against T4P4S [17], a DPDK-enabled software switch written in P4, and OpenvSwitch (OVS) [18], a DPDK-capable production quality open source software switch. For brevity, we only include the results for L2-FWD in Table II, where one can observe that MACS outperforms the other two switches in case of each core setting, and what is more, MACS scales better than T4P4S and OvS in terms of throughput while increasing the number of cores from 4 to 8. On the other hand, in case of ODP (i.e., MACS), special attention is needed for the CPU core affinity setting when exploiting the NUMA architecture (e.g., using more cores than 12 in our testbed) as automatic CPU core pinning (w/o explicitly defined) ends up assigning cores from other NUMA nodes, resulting in remote memory accesses and the consequent performance hits.

[10]and applies QoS policies: feature not currently implemented.

[11]Again, due to the GRE headers, the smallest packet size in UL is 82B.

TABLE II
THROUGHPUT (MPPS) FOR L2-FWD (100 ENTRIES, 64B) ON (INTEL XEON E5-2680 V4 @ 2.40GHZ, 100G NIC, 192GB RAM)

Datapath Switch	No. of Cores			
	1	2	4	8
MacS	9.42	18.36	30.75	36.72
T4P4S	8.20	16.30	28.10	29.70
OvS	6.20	12.20	21.30	22.90

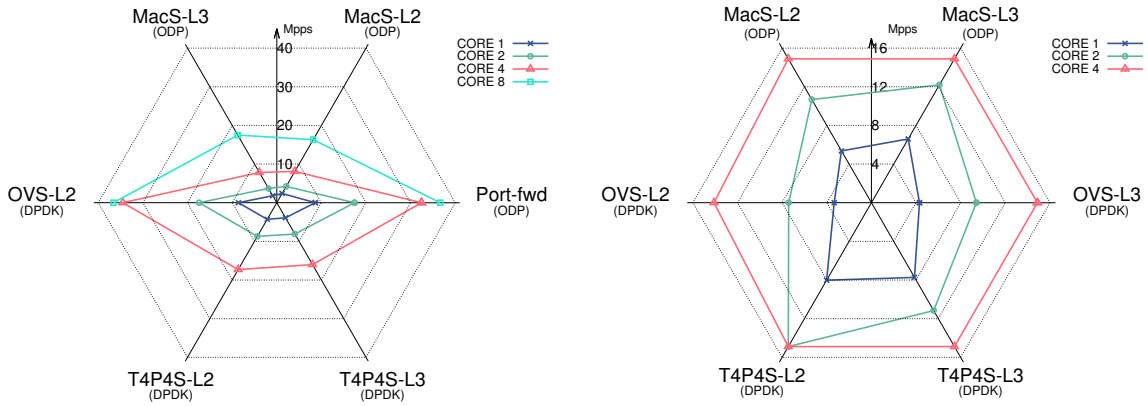
Portability. Achieving high performance from commodity-of-the-shelf (COTS) servers – a tenet of NFV – is challenging despite advances in I/O acceleration (e.g., DPDK) as the presence of multiple abstraction layers (e.g., hypervisor, libraries) prevent to access all hardware capabilities (e.g., CPU, NIC). Therefore in order to assess portability, we expand the evaluation to multiple platforms, in particular to the AARCH64-based Cavium R150-T62 (48 cores at 2.0 GHz, shared L2, no L3 cache, and 40G interfaces) while comparing performance of MACSAD against T4P4S and OVS.

The radar chart in Fig. 10a shows the measured throughput (Mpps) attained on the Cavium platform for the L2-FWD and the L3-IPv4 use-cases implemented via MACS (top), OVS and the base line ODP port forward application (middle), and T4P4S (bottom) when using 1 (blue), 2 (green), 4 (red) and 8 cores (cyan), respectively.

In our results for the AARCH64 architecture, DPDK-based switches perform better than their ODP-based counterparts because the Cavium switch only supports the first and already outdated version of ODP. To assess the raw performance capabilities, we measure the baseline ODP performance with the port-fwd application, which does nothing but forward packets from one port to the other without any table lookup (right hand side of Fig. 10a). The maximum throughput with one core is about 8.6 Mpps, around 20% less than the DPDK reference throughput (11.2 Mpps, not shown in the figure). In fact, ODP is only nearly equal to OVS-L2, which does one table lookup too. Due to the raw performance differences between ODP and DPDK in Cavium, the comparison cannot be considered fair but nevertheless serves to illustrate how the performance scales with increasing number of cores.

Both T4P4S and MACS show performance drop of about 33% against their baseline results of ODP and DPDK. We believe that with optimized new ODP support, MACS performance shall be on par with T4P4S. Current numbers shows that for the L2-FWD and L3-IPv4 use cases, T4P4S outperforms MACSAD in each case around 40% on average. But during the core scalability evaluation, T4P4S failed to run with 8 cores, whereas the MACS prototype easily exploited the available resources, e.g.,MacS-l2fwd throughput increased from 2.4 Mpps (1 core) to 16.3 Mpps (8 cores).

When comparing MACS with OVS and T4P4S in the same testbed (see Section IV) with 100 FIB table size, as shown in Fig. 10b, we observe that, for 1 core, T4P4S performs better reaching around 8 Mpps for L2-Fwd and L3-Fwd. With 2



(a) AARCH64 (b) x86_64

Fig. 10. Performance evaluation and comparison of different platforms and switches for selected use cases (100 entries per table) and varying CPU cores.

cores, T4P4S reaches line rate for the L2-Fwd whereas OVS L2 managed little less than 8 Mpps, whereas MACS behaves better (12 Mpps) for L3-Fwd with 2 cores. We note that all the platforms saturate the link when using 4 cores.

G. Adaptive Scalability by Dynamic CPU Core Allocation

While more cores improve performance, in case of over dimensioning, CPU core pinning and fixed allocation to packet processing can be considered as a waste of resources. We now investigate the feasibility of a proof of concept technique to provide dynamic CPU scaling through run-time (de)allocation of CPU cores to the packet processing tasks, i.e., ODP worker threads in case of MACSAD. Out of scope remains the decision of scaling up/down, which could be adaptive based on system load and/or performance measurements depending on traffic workload or other factors (e.g., energy consumption). Such an adaptive behaviour would make the system more efficient, especially in a multi-tenant environment, where de-allocated CPU cores could be used for other tasks.

The adaptive CPU scaling technique under evaluation consists of dynamically setting the number of RX queues and accordingly scaling up or down the number of cores. To scale down, MACS removes core-queue associations, releasing the core for kernel usage and leaving the RX queue without a descriptor.[12] Similarly, to scale up, MACS seamlessly acquires more cores and assigns them to the RX queues. For the proof of concept experiment, we use 4 cores (`A`, `B`, `C`, `D`) and 4 RX queues, and run with L3-IPv4 use case and different FIB sizes (100, 1K, 10K, 100K). We set MACS to start with 1 core (`A`) and after every 30 secs a new core is allocated (`B`, `C`, and `D`, respectively). After reaching the maximum core configuration (i.e., 4), MACS starts releasing cores, again in 30 secs interval.

Fig. 11 shows how the obtained throughput increases and decreases in line with the number of active cores. While only one experiment snapshot for each FIB size is presented, the observations were consistent over different runs.

The obtained results found evidence for an unexpected outcome regarding the performance behaviour of Receive Side Scaling (RSS) when only 100 different flows are balanced through the cores. Fig. 11 shows that the CPU core set (`AB`) achieves lower throughput compared to (`CD`). Since the sending rate was fixed throughout the experiment, the only explanation is the RX queue receiving less traffic over (`AB`) compared to (`CD`) core set and not a limitation of MACS TX queues. Under an ideal traffic distribution with both of the two-queue/two-core sets, we should observe the same throughput as in case of the (`ABC`) compared to (`BCD`) allocation, or when only cores `A` and `D` are used. The unequal flow distribution observed could be explained by specificities of the RSS hashing function implementation and the statistical nature of such a load-balancing mechanism and hence the challenge of always deciding on the optimal number of CPU cores for a certain throughput requirement for the target platform.

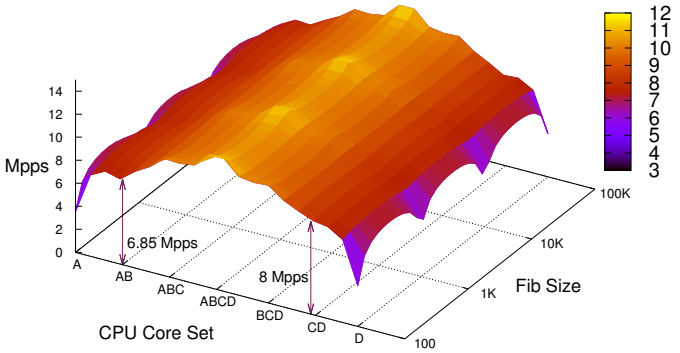


Fig. 11. Performance (Mpps) when dynamically (30s intervals) changing the sets of CPU cores allocated to packet processing for different FIB sizes.

As a second unexpected outcome, we detected an issue with some NICs (e.g., Intel 82599, X540) and the RX queues. In particular, when an RX queue is not fully flushed before removing its RX descriptors, the NIC stops processing packets altogether from all RX queues, whereas other NICs (e.g., Intel XL710 Fortville) did not have this limitation. We have reported this issue to the ODP community.[13]

[12]Note, a descriptor can exist even if the corresponding queue does not.

[13]https://bugs.linaro.org/show_bug.cgi?id=3618

V. Related Work

We now review some recent software switches and emphasize the relation to our proposed MACSAD framework. OVS provides relatively flexible processing but inherits performance limitations of the operating system's data plane and programmability cannot go beyond OpenFlow. The former can be addressed by recent "kernel-bypass" frameworks, e.g., DPDK, while the latter can be addressed by work like PISCES [19] to add P4-based flexible packet parsing support but limited data plane reconfigurability due to OVS pipeline construction. OpenSwitch (OpenSwitch)[14], and Netronome Network Flow Processor (NFP)[15] offer platform-limited P4 support.

Most closely related work is the T4P4S [17] software switch based on a Hardware Abstraction Layer (HAL) incorporating P4 to DPDK mapping. MACSAD "Transpiler" module is based on the T4P4S code. T4P4S, however, is not multi-platform and DPDK's LPM for IPv6 is not implemented, in addition to the lack of validation of complex pipelines due to limited P4 primitives support.

The project DC [20] provides P4 based VXLAN implementation designed for data centers facing similar challenges as MACSAD. Dietz et al. [21] present a BNG implementation using an alternate modular framework called Click. It uses tiny Virtual Machines (VMs) with fast boot and small memory footprint, and an excellent candidate for NFV adoption. The results reported are also very similar to MACSAD for larger packets sizes. However, MACSAD differs in multi-platform support and P4 based data plane definition.

VI. Conclusions and Future Work

As an amalgam of the protocol-independent P4 programming language and the ODP platform-independent SDK [6], MACSAD offers an SDN data plane design approach capable of supporting complex pipelines such as DCG, BNG with portability and performance. While we showed the length of programmability with $P4_{16}$ support and the implemented use cases released to the public domain, the breadth of portability is demonstrated by running MacS over different platforms (x86_64, AARCH64) and packet I/O drivers (Netmap and DPDK). The performance results match the state of the art efforts on the leading OVS and P4-based software switches like T4P4S. To investigate scalability, we obtained results for platforms with multiple CPU cores and interface speed (10G, 40G, 100G) suggesting that MACSAD fares well against OVS and T4P4S. The adaptive CPU scaling featured technique offers a novel approach applicable to many-core software switch designs and opens promising research opportunities.

The public code release of MACSAD is one of our open source contributions. We also released BB-Gen to easily create test traffic and the required trace files for more than 1M entries and different header definitions required by the P4 use case. Various PCAP and trace files have been contributed to the NFPA repository. Our IPv6 lookup mechanism will be proposed for adoption in ODP.

[14]https://www.openswitch.net

[15]http://open-nfp.org

We will continue to improve $P4_{14}$ and $P4_{16}$ support by implementing *variable width header field support*, *stateful datatype support* and *ternary table lookup support*, and, in turn, support more complex pipelines. The idea on adaptive CPU scaling will be further explored potentially in combination with SDN controller feedback loops and core utilization measurements to develop new run-time core allocation algorithms. We are also planning to add support for multiple MACSAD instances serving different pipelines and to investigate deeper about latency, packet loss and cache utilization.

Acknowledgment

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreements UNI.61 and UNI.63 We thank the P4@ELTE team from Eötvös Loránd University, Budapest, Hungary for the results of the OVS and T4P4S use cases for AARCH64.

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] 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.

[3] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *PAM*, 2015, pp. 347–359.

[4] L. Csikor, L. Toka, M. Szalay, G. Pongrácz, D. P. Pezaros, and G. Rétvári, "HARMLESS: Cost-Effective Transitioning to SDN for Small Enterprises," in *Proceedings of IFIP Networking*, 2018.

[5] P. Bosshart et al, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, Jul. 2014.

[6] Opendataplane. [Online]. Available: http://www.opendataplane.org

[7] 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.

[8] P. G. Patra, C. E. Rothenberg, and G. Pongracz, "Macsad: High performance dataplane applications on the move," in *IEEE HPSR*, 2017.

[9] Intel, "Dpdk: Data plane development kit," http://dpdk.org.

[10] L. Rizzo, "netmap: A novel framework for fast packet i/o," in *USENIX ATC 12*.

[11] F. Rodriguez, P. G. Patra, L. Csikor, C. Rothenberg, P. Vörös, S. Laki, and G. Pongrácz, "BB-Gen: A Packet Crafter for P4 Target Evaluation," in *ACM SIGCOMM'18 Demo and Poster Session*, 2018.

[12] L. Csikor, M. Szalay, B. Sonkoly, and L. Toka, "NFPA: Network function performance analyzer," in *IEEE NFV-SDN*, 2015, pp. 17–19.

[13] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices," RFC 2544, 1999.

[14] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of Frameworks for High-Performance Packet IO," ser. ANCS '15. Washington, DC: IEEE Computer Society, pp. 29–38.

[15] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," *ACM SIGCOMM*, pp. 191–202, 1998.

[16] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic routing encapsulation (gre)," Internet Requests for Comments, RFC 2784, 2000.

[17] 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 Posters and Demos*, 2016.

[18] 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 *USENIX NSDI*, 2015.

[19] M. Shahbaz et al, "PISCES: A Programmable, Protocol-Independent Software Switch," in *ACM SIGCOMM*, 2016.

[20] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, "DC.p4: programming the forwarding plane of a data-center switch," *SOSR*, vol. 4, pp. 1–8, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2774993.2775007

[21] T. Dietz, R. Bifulco, F. Manco, J. Martins, H. J. Kolbe, and F. Huici, "Enhancing the bras through virtualization," in *Proceedings of the 2015 1st IEEE NetSoft*, April 2015, pp. 1–5.

P Gyanesh Kumar Patra is a Ph.D. candidate at University of Campinas, Brazil and works with Information & Networking Technologies Research & Innovation Group (INTRIG). His primary interests are in SDN and programmable dataplane. He has nearly five years of industry experience working on network operating systems, data center protocol like QCN, SPB, etc. He also worked as a visiting researcher at Ericsson Research, Hungary. Currently, he is pursuing his thesis on building cross-platform software switch for programmable dataplanes.

Fabricio E Rodriguez Cesen is a MSc. student at University of Campinas, Brazil and works with Information & Networking Technologies Research & Innovation Group (INTRIG). He holds the Networks and Data Communication Engineering degree from the Army University - ESPE of Ecuador. His principal interests are in SDN, forwarding models, and P4 language. Currently, he is pursuing his thesis on IPv4/v6 LPM implementation for programmable dataplanes.

Juan Sebastian Mejia. received the degree in Electronic and Telecommunication engineering from the University of Cauca, Popayan, Colombia, He is currently pursuing the M.Sc. degree in electrical and computer engineering with the University of Campinas (UNICAMP) and work in Information & Networking Technologies Research & Innovation Group (INTRIG). His research focus is programmable dataplanes, P4 language and BRAS/BNG network device. He is also interested in SDN, NetFPGA.

Daniel Lazkani Feferman is currently a researcher at the Information & Networking Technologies Research & Innovation Group (INTRIG) and a MSc. Student at University of Campinas (UNICAMP), holds a BSc. in Telecommunications engineer from Fluminense Federal University (2017). From 2015-2016 he was an exchange student at the New York Institute of Technology (NYIT). His main interests include SDN, P4 language, VxLAN protocol and privacy.

Levente Csikor is a post doctoral researcher at Budapest University of Technology and Economics (BME), where he has received his M.Sc. and Ph.D. degree from in 2010 and 2015, respectively. He has been a research associate at the School of Computing Science, University of Glasgow working on an Engineering and Physical Sciences Research Council (EPSRC) funded project in 2017. His interests are focused on the dataplane performance of different software-based network functions, troubleshooting, orchestration and Denial of Service attacks.

Christian Esteve Rothenberg is an Assistant Professor in the Faculty of Electrical & Computer Engineering (FEEC) at University of Campinas (UNICAMP), Brazil, where he received his Ph.D. and currently leads the Information & Networking Technologies Research & Innovation Group (INTRIG). His research activities span all layers of distributed systems and network architectures and are often carried in collaboration with industry, resulting in multiple open source projects in SDN and NFV among other scientific results.

Gergely Pongracz is an expert in Ericsson Research in the area of programmable dataplane. He graduated at the Technical University of Budapest in 2000. In 2004 he became a research engineer at Ericsson. Recently he is working on NFV and SDN topics, especially in the programmable networking area. These projects resulted in well received papers and demos, such as a paper on IEEE SigComm in 2016 or demos at the Mobile World Congress in 2015 and 2017.