Towards Realization of High Performance Programmable Datapaths using Domain Specific Language

P Gyanesh Kumar Patra, Christian Esteve Rothenberg (Advisor)

Department of Computer Engineering and Industrial Automation (DCA)
School of Electrical and Computer Engineering (FEEC)
University of Campinas (Unicamp)
CEP 13083-852 – Campinas, SP, Brasil

{gyanesh,chesteste}@dca.fee.unicamp.br

Abstract – Network virtualization has paved the way for renewed interest in programmable data plane research. Programming Protocol-Independent Packet Processors (P4) being a Domain Specific Language (DSL) brings rich abstractions through a target-independent high-level programming language that can be compiled to run on a variety of targets (e.g., ASIC, FPGA, GPU). We present our Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) proposal in this article which blends P4 network abstractions with low-level but cross-platform (HW & SW) APIs brought by OpenDataPlane (ODP) and thereon supports different vendors and architectures. We explain the current open questions and trends, and show how MACSAD fits into the landscape. The design decisions behind MACSAD architecture and their impact towards attaining the required characteristics, namely, programmability, performance, portability, and scalability are investigated. Furthermore, We present the results for dynamic CPU core (re)allocation feature and impact of compiler optimization technique exploiting memory-level parallelism in MACSAD.

Keywords – MACSAD, P4, DSL, OpenDataPlane, Software-Defined Networking, Programmable Dataplane

1. Introduction

Evolution trends in networking devices are largely performance oriented while supporting multitude of protocols and applications. It relies on closed 3-5 years slow development cycle by the vendors improving the tightly integrated software and hardware. But in recent times we witnessed the surge of two trends having significant impact on networking and it’s evolution. First the arrival of server and storage virtualization forced network professionals to rethink networking devices also as virtualized devices instead as standalone devices only. Then the advent of cloud pushed networking industry towards virtualization and put central control as a new requirement. These paved the way for the new paradigm of disaggregation of control plane functions from data plane functions in networking devices and found itself adopted by OpenFlow protocol [4] which popularize match + action abstractions aimed to be the de facto standard for defining programmable data planes. Hardware disaggregation as a result of virtualization, and control plane separation brought forward new possibilities for data plane development such as generic data plane model, data plane programmability, unified domain specific language etc.. Our research is focused on data plane aspects which unlike control plane have been less fortunate in terms of advancements due to it’s closed nature and heterogeneous contributing community. Ongoing trends in our scope towards deep programmable data plane include:

1. Designing protocol independent programmable hardware using match + action abstractions as pursued by Protocol Independent Switch Architecture (PISA) [10] and RMT designs.
2. Top-down approach providing target-agnostic data plane programmability using high-level abstractions of a unified DSL. Our choice of DSL candidate is P4 [9, 12].
3. Bottom-up effort bringing portability using platform-agnostic low-level Software Development Kits (SDKs) for datapath. We explored OpenDataPlane (ODP) [7] open source project as a candidate offering vendor-neutral abstract APIs covering common features across several targets.
4. New protocol to define SDN-centric northbound interfaces using a configuration file. The emerging P4Runtime (PRT) from P4 consortium is a step towards this as PRT server can create the northbound Application Programming Interfaces (APIs) at runtime to communicate with remote or local controller.
5. Generic communication protocol for control plane to configure and manage the underlying protocol independent and programmable data plane. Here, the PRT Client (not server as in last point) is responsible to provide the necessary functionalities.
6. Ability to define and configure a data plane at runtime by means of a configuration file agnostic to the underlying hardware and supported protocols. The configuration file format defined by PRT is a perfect choice for this as it is capable of defining the data plane and also proving the configuration details to the control plane of SDN controller.

We explore these points while explaining the design and evaluation of our proposed MACSAD project, a high performance data plane, which blends the high-level protocol-independent programmability of P4 with low-level but cross-platform (HW & SW) APIs brought by OpenDataPlane (ODP) approaching portability and multi-architecture support. We will show our sweet spot approach with four characteristics, namely, performance, portability, programmability and scalability. We will also present our findings on dynamic CPU core (re)allocation feature and optimization activity for MACSAD followed by brief description of future activities and conclusion.

2. MACSAD

SDN paradigm acknowledges the “match + action” abstraction to define data planes in support of relaxed table definition where size, number and functionality of tables are programmable. This new paradigm of programmable targets are achievable conjointly with a generic data plane model e.g., PISA as in Fig. 1 and a unified DSL e.g., P4 [9].

We aim to define programmable targets confirming PISA with our proposed MACSAD compiler system by bringing P4 to low level abstraction using ODP APIs. MACSAD high-level architecture consists of multiple modules in sought for Protocol Independence and Target Independence.

2.1. Auxiliary Frontend

It is responsible for transforming P4 into Intermediate Representation (IR) required by ‘Core Compiler’. Being both P414 [9] and P416 [12] compatible, it generates High Level Intermediate Representation (HLIR) [11] using p4c or JSON IR using p4c compiler respectively. The top left rectangle in Fig. 2 depicts the IR passed on to the Transpiler submodule.

2.2. Auxiliary Backend

The Auxiliary Backend represents a common SDK comprising of all internal and helper APIs based on ODP APIs to support P4 abstractions which can also leverage the hardware acceleration and optimization features of the target when possible.

2.3. Core Compiler

Being responsible for creating target image from the input IR, it houses the Transpiler and Compiler submodules.

a) Transpiler. It is our source-to-source compiler to auto-generated code in ‘C’ for packet parsing and packet flow across the tables defined in the P4 program, and feed the auto-generated code to the Compiler submodule.

b) Compiler. This is the final stage of MACSAD. It generates MACS using 2.2 and output of 2.3 with the underlying GCC/LLVM compilers.

2.4. Analysis & Evaluation

Programmability of MACSAD can be affirmed by various use cases presented in [14] [2] [3]. Fig. 3 shows throughput comparison among the use cases with increasing complexity in terms of number of table lookups and packet actions. Various use cases supported by MACSAD are diverse and include L2-FWD, L3-IPv4/v6, Data Center Gateway (DCG), Broadband Network Gateway (BNG), Network Address Translation (NAT) as shown at

---

1.https://github.com/p4lang/p4-hlir

---

The MACSAD compiled binary code is referred to as MACSAD Switch (MACS) throughout the text
XI Encontro de Alunos e Docentes do DCA/FEEC/UNICAMP (EADCA)
XI DCA/FEEC/University of Campinas (UNICAMP) Workshop (EADCA)

Campinas, Brazil, November 08-09, 2018

As explored here [17], more than 70% of packet processing time is spent on table lookup in the datapath which confirm our results for NAT, DCG and BNG use cases which implement increasing number of table lookups and tunneling support.

Performance & Scalability of MACSAD are compared against two related works (such as the P4 based switch T4P4S [6] and the DPDK-capable production quality open source software switch OpenvSwitch (OVS) [16]) as shown in Table 1 for L2-FWD use case. The result shows that MACS outperforms the other two switches in case of each core setting, and also scales better than T4P4S and OvS in terms of throughput while increasing the number of cores.

Portability of MACSAD has allowed to bring the use cases to x86, ARMv6, and ARMv8 platforms spanning Intel Servers, Raspberry Pi2, and Cavium switches and is explored in detail in our previous research work at [14].

3. Dynamic CPU Core Allocation

We present here the feasibility of a novel technique providing dynamic CPU scaling through run-time (de)allocation of CPU cores in MACSAD. Scaling up/down can be adaptive based on system load, on traffic workload or other factors (e.g., energy consumption). Such behaviour is extremely useful in multi-tenant environment, where de-allocated CPU cores could be used for other tasks. For this 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 time the timer (30 secs) expires a new core is allocated (B, C, and D, respectively). Similarly, MACS also starts releasing cores from the maximum core configuration (i.e., 4) after each 30 secs interval. Fig. 4 shows how the throughput increases and decreases in line with the number of active cores.

4. Compiler Optimization

MACSAD implements a number of optimization techniques in every steps and modules. But our work towards exploiting the memory-level parallelism between CPU and main memory [1] needs attention for it’s impact on performance by targeting the memory-bound steps of packet processing i.e., the table lookup which consists of table key creation and the actual lookup step. We observed that different lookups can be similar, and hence we implemented batched table key creation and table lookup to exploit the memory level parallelism while hiding the CPU-memory latency. Table 2 shows the initial improved throughput results for L2-FWD use case.

5. Conclusion

Pisces [8] was one of the first [3] switch based on OVS [16] but it was limited by OVS abstractions. T4P4S [6] is close related work under active development and is a part our state of the art evaluation. Looking at the current landscape, we be-
Table 2: Throughput (Gbps) for L2-FWD (100 entries, 64B) on (Intel Xeon E5-2620v2, 10G NIC, 64GB RAM)

<table>
<thead>
<tr>
<th>No. of Cores</th>
<th>Without Batching</th>
<th>With Batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.8</td>
<td>3.0</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>5.7</td>
</tr>
</tbody>
</table>

lieve we have great opportunity with our MACSAD offering which brings P4 DSL support over multiple target platforms. We have shown that MACSAD excelled in programmability, portability, performance and scalability compared to other related works. We are working to bring more compiler optimization techniques for performance improvements focusing on memory-bound and CPU-bound steps separately while integrating our findings into the core compiler to auto-generate code instead of working with static code. We are also currently evaluating P4 Runtime (PRT) and looking forward to support it in the near future.

Acknowledgment

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreements UNI.61 and UNI.63.

References


