Network Address Translation using a Programmable Dataplane Processor

Juan Sebastian Mejia Vallejo[1] **, Daniel Lazkani Feferman**[1] **,**
Christian Esteve Rothenberg[1]

[1]Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101, 13083-970 – Campinas, SP, Brasil

{jmejia,fefer,chesteve}@dca.fee.unicamp.br

Abstract. *A short-time solution for problems facing the Internet of IP address depletion and scaling in routing is the address reuse solution placing Network Address Translators (NAT) at the borders of stub domains. In this article, we propose an implementation of NAT using Programming Protocol-Independent Packet Processors (*P4*) language, taking advantage of its features such as target-agnostic dataplane programmability. Through the MACSAD compiler, we generate a software switch that achieves high performance with for different hardware (*H/W*) and Software (*S/W*) platforms. The main contributions of this paper relate to the performance evaluation results of the NAT implementation using P4 language with MACSAD compiler.*

1. Introduction

Considering the continuously grown of Internet services such as Voice Over IP, Multimedia Over IP [Schaar and Chou 2007], on-line games and the progressive depletion of public IPv4 addresses, the Network Address Translation (NAT) [Egevang and Francis 1994] became the primary method to allow multiple Private IPs to get access to the Internet through a limited number of Public IP. Typically, an extensive Private Network consists of a range of private IP (e.g., 10.0.0.0 to 10.255.255.255), enabling the communication of servers, printers, etc. However, the number of public IPs is limited and costly. In recent years, an initiative of turning rigid hardware-based networks into software-based emerged named as Software-Defined Networking (SDN), splitting the control and dataplane functions to turn some network functionalities into virtualized software devices running on servers (e.g., off-the-rack x86 servers), switches or even cloud computing infrastructure [Han et al. 2015].

OpenFlow is one of the first projects that follows the same SDN methodology by letting the administrators to define dataplane functionalities and is currently maintained by ONF (Open Networking Foundation). However, after more than 10 years since its foundation, the interface continues to have hardware compatibility issues and slow support of new packet headers.

In the recent development, the P4 [Bosshart et al. 2014] language introduces dataplane programmability by allowing multiple targets to use the same Domain Specific Language (DSL). MACSAD framework delivers cross-platform portable dataplane applications, it brings performance, flexibility and portability in dataplane. Thus, the P4 NAT S/W

switch uses MACSAD and familiar concepts of SDN and Network Function Virtualization (NFV) dataplanes [Niu et al. 2016] to achieve a high performance programmable NAT switch. We present the NAT performance evaluation varying either target platform parameters (e.g., Packet I/O, number of cores) and packet size. The rest of this paper is structured as follows: Section II provides the background details. Section III describes the Methodology. Section IV expose briefly the Related work. Finally, Section V discusses conclusion and future works.

2. Background

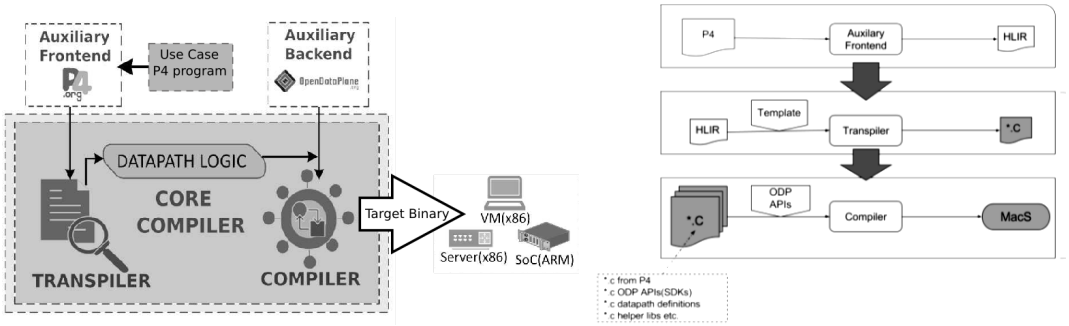
2.1. Programming Protocol-Independent Packet Processors (P4)

Considering the development of OpenFlow (OF) protocol over the years, few limitations were found (e.g., most switches have multiple policies and stages of match+action tables, limited TCAM space, etc.). Furthermore, to include a new header on OF it was necessary to update its version with retro-compatibility, making the release of new versions too slow [Bosshart et al. 2014]. Initially, the first version of OF started with 12 fields. Today, the last version contains more than 40 fields and there are important headers that are not supported yet. These limitations led to P4 language define three main goals:
· **Reconfigurable in the field**: Redefine packet parsing and processing in the field even after it is implemented.
· **Protocol independence**: The switch should be configurable and not tied to a specific header format.
· **Target independence**: Packet-processing functionality should be independent of the target where it will be deployed. Hence the compiler could map to different forwarding devices.
Thus, P4 language is a high-level abstraction that supports different targets, based on S/W or H/W. The P4 uses the *match+action* table model. Through the language it is possible to achieve dataplane programmability, a P4 program is composed of five main components: tables, actions, parser, control, and headers.

2.2. Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD)

The MACSAD is a P4 compiler that focuses on high performance with portability and flexibility. As shown in Figure 1a, the MACSAD [Patra et al. 2016] is composed of three main modules (See Figure 1b) :

- **Auxiliary frontend:** responsible to aggregate several Domain Specific Language (DSL). It creates an Intermediate Representation (IR) from P4 code (both $P4_{14}$ and $P4_{16}$ versions); this representation will be used by the core compiler. In this module, the P4-hlir project is used to translate P4 programs to a High-Level Intermediate Representation (HLIR).
- **Auxiliary backend:** aims to give a common SDK, using OpenDataPlane (ODP) APIs [OpenDataPlane 2013] Furthermore, it contains developed libraries to bring support for P4 primitives.
- **Core compiler:** includes the transpiler and compiler modules. It merges the result of the frontend (the HLIR) and backend (the ODP APIs) to provide the binary which will be used by the device either by a Virtual Machine (VM), Raspberry (ARM), server (x86) or a SoC (ARM).



(a) Macsad architecture. Adapted from : [Patra et al. 2017]

(b) 3-Tier compilation process

Figure 1. MACSAD architecture.

Transpiler receives the result from the Auxiliary frontend and automatically generates the Data-path Logic codes. This tool is responsible for the definition of the size, queues, lookup mechanism, and type of tables that will be created using the target's resources. The group of ".c" files generated by the transpiler contains ODP APIs, helper libraries and parts of the P4 program.
The *Compiler* creates an executable S/W switch from the generated "C" code above mentioned, in our case a NAT and the controller interface to add the table entries. Currently, MACSAD uses Low-Level Virtual Machine and GNU Compiler Collection (GCC) compiler to guarantee the support of multiple targets.

3. Methodology

In this section, we explain the methodology and implementation, followed by the performance results.

3.1. Testbed

As depicted in Figure 3b Our testbed includes two servers with Intel Xeon E5-2620v2 processors (6 cores), 64GB of memory, running Ubuntu Linux 16.04 LTS (kernel 4.4) and two dual-port Intel X540-AT2 NIC (10G).
For the experiments, one node (Tester) runs a packet generator (NFPA) [Csikor et al. 2015] sending packets of different sizes to the other node with DUT via 10GB NIC. Packets are sent back through another NIC of the Tester to the first node. NFPA uses Pktgen-dpdk v3.4.5 [Turull et al. 2016] and DPDK v17.08 for packet generation. The DUT supports multiple packet I/Os with DPDK v17.08, ODP v1.16.0.0, and Netmap v11.2 versions.

3.2. NAT Dataplane P4/MACSAD Implementation

This use case has been implemented with $P4_{16}$, which is the latest P4 version, bringing new instructions to define tables, actions and controls. Figure 2 illustrates the implemented pipeline of our NAT P4 program[1], which is divided in Upload (UL) and Download (DL) data paths. The packet processing is supported with multiple sets of tables as described as follow:

[1] https://github.com/intrig-unicamp/macsad-usecases/blob/master/p4-16/nat.p4

- **Set interface:** it set up the network interface as external or internal to separate the UL and DL traffic.
- **L2:** The NAT acts as an L2 learning switch and processes ARP packets coming from the host; an entry is created (or updated) in the MAC address table.
- **NAT UL/DL:** Since users within a private IP network send packets to a public network, NAT is required to translate IPv4 address and TCP ports in both ways, this table store TCP port and IPv4 address entries to perform packet processing and forwarding. Otherwise, incoming packets without equivalent entries are dropped.
- **IPv4 routing:** The routing table stores the next hop based on the IP address. It is based on the Longest Prefix Match (LPM) implementation.

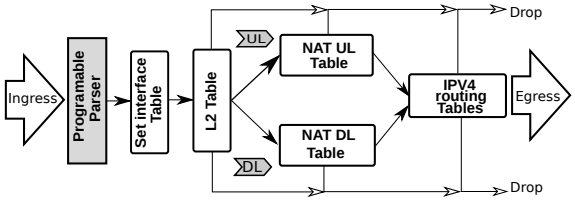


Figure 2. NAT pipeline

3.3. Performance Evaluation

Figure 3a depicts the NAT use case handling traffic between a Private and an External (Public) network, with the primary dataplane functions divided into a UL path coming from the *Host* (IP address 10.1.1.10) to an Internet *Server* (IP: 213.1.1.1), and DL path from the *Server* back to the *Host*.

UL: The UL traffic starts with *Host* that sends a packet destined to *Server*, when the packet arrives at our NAT software switch, the source IP is rewritten with the public IP and the following TCP port. Finally, the NAT performs IPv4 packet forwarding and select the output port to send the packet through the external network.

DL: As a response, the *Server* (IP: 213.1.1.1) sends TCP traffic back to the host (10.1.1.10) via NAT software switch through the external interface using NAT Public IP, which is converted to the Private IP by the NAT. Finally, it completes the IP packet forwarding by selecting the next hop and output port towards Host.

We have generated random traffic traces for both datapaths varying the source and destination IPv4, L2 source address, TCP source and destination ports inside the Forwarding Information Bases (FIBs). All these fields were filled through an SDN controller. We run MACSAD with three CPUs combinations (2, 4, 6 CPUs), FIB table of 100 entries, and three Packet I/O drivers (Socket-mmap, Netmap, DPDK). In the next subsection, we present the testbed setup and performance results.

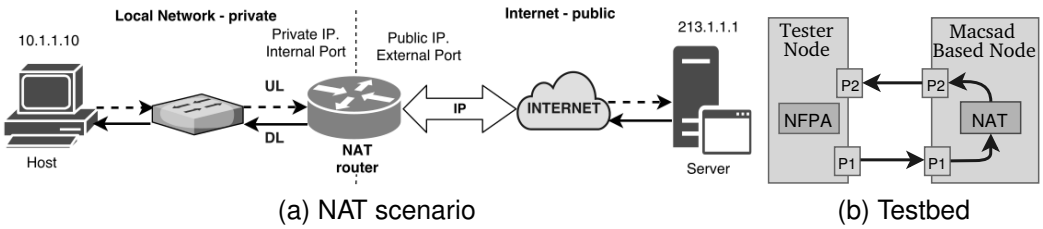
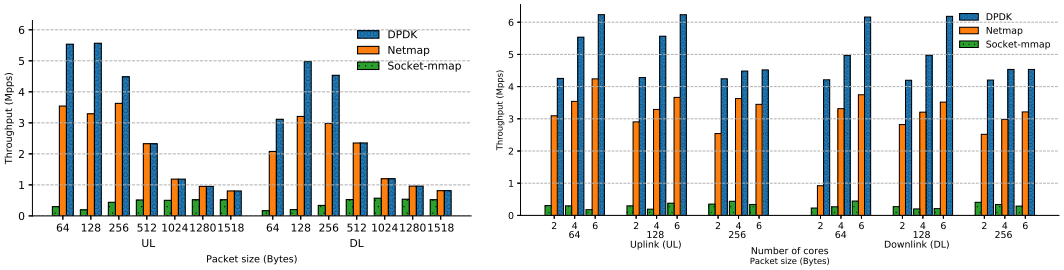


Figure 3. NAT Scenario.

3.4. Results

The bars in Figures 4a and 4b measure the performance on NAT with a packet flow of 100 entries. From the results, we observe that the best performance for all the packets sizes was with DPDK NIC driver, in fact, this PktI/O achieve line rate (i.e., 4,52 Mpps for a packet size of 256B) both UL and DL datapaths with medium and bigger packet sizes (256B-1518B). On the other hand, the worst case is clearly with 64B and 128B using Socket-mmap, as it handles more packets per second. However, this was already expected as for low sizes it needs to match more tables, impacting the results. In Figure 4b we observe the performance increase by adding CPU cores, this is mainly using notable on Netmap and DPDK NIC drivers, considering that more cores usually means more instructions can be done, we were already expecting these results. Also, due to queues type; unlike of Socket-mmap which use hardware queues, both DPDK and Netmap, can assign virtual queues to increase performance.



(a) NAT Performance comparison for different NIC drivers (4 cores, 100 table entries)

(b) NAT Performance comparison for different number of cores, drivers and packet sizes

Figure 4. NAT performance results.

4. Related Work

The work in [Hwang et al. 2015] describes a S/W router, with Layer 3 forwarding functionalities that reside in distinct VMs. They use NetVM platform obtaining throughputs of up to 10 Gbps. Rather of running into H/W limitations such as NIC, their implementation is limited by the available processing capacity. In [Patra et al. 2017] presents a small L2 S/W switch as MACSAD proof-of-concept on which the dataplane is created with P4 (P4_14 version) which is running with MACSAD for different PktI/O drivers. In [Roberto et al. 2013] we found an approach to virtualize a Broadband Remote Access Server (BRAS) based on Click OS, a tiny Xen virtual machine designed specifically for network processing it can achieve line rate of 10Gbps with Netmap and VALE PktI/O drivers. The work in [Bondan et al. 2014] presents the management requirements in the context of a specific NFV enabler platform called ClickOS; they propose a network scenario with a NAT function. However, they studied the requirements to deploy the virtualized devices to ease the adoption of NFV by network operators.

5. Conclusions

In this article, we presented a brief description of the NAT S/W dataplane using P4 language; a new language that aims to revolutionize networks by giving programmability to

the dataplane.
We exposed a NAT S/W switch running on MACSAD over x86 platform and different PktI/O drivers (DPDK, Netmap, Socket-mmap), achieving 10G line rate with medium and large packets, confirming his approach of support complex SDN dataplanes with portability and high performance. As a future work, we intend to implement and show the performance results in other platforms such as VMs and ARM systems. Our results encourage us to identify and resolve the bottlenecks mainly on smaller packets size and consider more performance metrics as processing delay, CPU cycles, etc.

References

Bondan, L., d. Santos, C. R. P., and Granville, L. Z. (2014). Management requirements for clickos-based network function virtualization. pages 447–450.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication*.

Csikor, L., Szalay, M., Sonkoly, B., and Toka, L. (2015). Nfpa: Network function performance analyzer. *IEEE Conference on Network Function Virtualization and Software Defined Networks Demo Track*.

Egevang, K. B. and Francis, P. (1994). The ip network address translator (nat). RFC 1631, RFC Editor. `http://www.rfc-editor.org/rfc/rfc1631.txt`.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Hwang, J., Ramakrishnan, K. K., and Wood, T. (2015). Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, 12(1):34–47.

Niu, Z., Xu, H., Tian, Y., Liu, L., Wang, P., and Li, Z. (2016). Benchmarking nfv software dataplanes. *CoRR*, abs/1605.05843.

OpenDataPlane (2013). OpenDataPlane.org. `https://www.opendataplane.org`. Accessed: 2018-01-30.

Patra, P., Rothenberg, C., and Pongracz, G. (2017). MACSAD: High performance dataplane applications on the move. *IEEE International Conference on High Performance Switching and Routing, HPSR*, 2017-June.

Patra, P. G., Rothenberg, C. E., and Pongrácz, G. (2016). MACSAD: Multi-Architecture Compiler System for Abstract Dataplanes (Aka Partnering P4 with ODP). In *ACM SIGCOMM'16 Demo and Poster Session*.

Roberto, B., Thomas, D., H, F., A, M., M, J., N, S., and K, H.-J. (2013). Rethinking Access Networks with High Performance Virtual Software BRASes. *EWSDN*.

Schaar, M. v. d. and Chou, P. A. (2007). *Multimedia over IP and Wireless Networks: Compression, Networking, and Systems*. Academic Press, Inc., Orlando, FL, USA.

Turull, D., Sjödin, P., and Olsson, R. (2016). Pktgen: Measuring performance on high speed networks. *Computer communications*, 82:39–48.