UNICAMP

Juan Sebastian Mejia Vallejo

Architecture and implementation of a Broadband Network Gateway using a programmable dataplane processor

Campinas

2018

UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Juan Sebastian Mejia Vallejo

Architecture and implementation of a Broadband Network Gateway using a programmable dataplane processor

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Supervisor: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Este exemplar corresponde à versão final da tese defendida pelo aluno Juan Sebastian Mejia Vallejo, e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

Campinas
2018

Inclua aqui o pdf com a ficha catalográfica fornecida pela BAE.

Inclua aqui a folha de assinaturas.

Abstract

Broadband Network Gateways (BNGs) play a crucial role in today's networks, it hands all access network traffic (e.g., DSL traffic), provisioning of such policies like rate limiter, tunneling, and complex network management mechanisms that a Internet Service Providers (ISPs) needs to implement his services. These network devices are expensive, proprietary, limited and slow device upgrading, become a point of failure to deploy new features, add functionalities and correct issues on the network without disrupting the normal service operation.

In this work, we propose a software-based virtualized BNG device running on inexpensive commodity hardware using high-level language for programming protocol-independent packet processors over a Multi-Architecture Compiler System to remove some of these barriers to innovation.

Keywords: Computer Networks; Software Defined Networking; OpenFlow; Future Internet.

Resumo

Adicionar resumo em portugues...

Palavras-chaves: Redes de Computadores, Software Defined Networking, OpenFlow, Internet do Futuro.

Contents

When a goal is reached, it is necessary to think and value those things that led us to reach it; the walked path and the learned lessons. I would like to dedicate this work all the people who gave me their words of encouragement, wisdom, love, and support; this goal has been possible thanks all of you. I am sure that in any place God will allow us to meet people to keep growing in both personal and professionally.

Acknowledgements

My thanks to my parents, whoever has motivated me to go further and give me the best example of dedication.
I am grateful for my family, for his support in the distance and send me his best wishes and prays.

Christian Esteve Rothenberg, my supervisor, and friend. Many thanks to INTRIG group for bringing me support many times even for its friendship outside of the university.

Ericsson Innovation Center Brazil, for sponsoring this work, and Ericsson Traffic Lab in Hungary, for technical support.

“…”

List of Figures

List of Tables

1 Introduction

The advent of new services like Video-on-Demand (Vod), video conference, Virtual Private Network (VPN) and cloud-based new services, has increased the demand for access to broadband services (The Organization for Economic Co-operation and Development (OECD), 2016). In addition, many further access technologies such as xDSL, optical access and wireless technologies such as WiMAX and LTE require rapid deployment of services and devices guaranteeing performance in Internet Services provider (ISP) networks.

In an Internet service provider (ISP) network the Broadband Network Gateway (BNG) has the function of managing all access network traffic (e.g., DSL traffic) and other critical functions like to allow access and authentication for thousands of subscribers, monitoring, establishing sessions, tunnels and controls the user line rate. The fact that all sessions tunnels (e.g.., PPPoE, GRE) are terminated at the BNG means that is aggregated in a single point causing poor performance and quality of service. Therefore is no surprise that this device becomes expensive and hardware proprietary boxes, often the operator pay for some functionality that won't be used and the hardware boxes upgrade require a long wait until the next version available.

In recent years, in order to resolve this problem to turn this rigid hardware device into a software-based network and reduce time to market of new services and those functionalities have been decomposed and dynamically instantiated at different points of the network. This concept is following the trend of Software-Defined Networking (SDN) and Network functions virtualization (NFV) that turn some network functionalities into virtualized software processing running on a server (e.g., off-the-rack x86 servers), switches or even cloud computing infrastructure (HAN *et al.*, 2015).

Our approach to create an fully open and programmable BNG data plane is using a high-level language for programming protocol-independent packet processors (P4), this is a Domain-specific language (DSL) with a number of functions optimized around network data forwarding. Such a DSL can support customizing the forwarding behavior of the switch and may also be ported to other hardware or software switches that support the same language. In this work, we present an approach to implementing a BNG software switch implementation, and it is built on top of the MACSAD Switch target and P4 language that can provide dynamic and flexibility to the service provider to optimize the traffic on the network.

We discuss the architecture and protocols to deploy and concepts around the BNG software switch.

The rest of this document is structured as follows: Section II provides the background and related work. Section III The problem statement and objectives, Section IV and Introduces the proposed design and some results, Section V Working plan and Execution schedule.

Objectives

To address the identified issues, the main objective of this work is to design, implement and evaluate the BNG software switch in a Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD). To this end, the following specific objectives are proposed:

For each of the issues mentioned in section **??**, we have some specific objectives in order to take action to those problems:

- Designing the architecture and functional implementation of a BNG software switch. Adding the data plane functionalities with the latest version of P4 lang and compiling our BNG p4 program in an emulation environment in order to test our P4 implementation.
- Defining and implementing P4_16 support for MACSAD. The MACSAD core works with C code in order to compile and run the software switch at execution time over a commodity server. According to the above, we need to implement the HLIR16 project and his dependencies such as P4C compiler to create an intermediate representation that brings data structure used internally in the MACSAD core to translate P4 into C.
- Integrating the BNG implementation generated with MACSAD to support the Traffic Manager ODP API. It adds QoS support to the BNG software switch. On the other hand, the controller has to interact either with MACSAD to manage table actions management as with the Traffic Manager block hence; this interface will be added as well.
- Performance evaluation with NFPA tool. Configuring a real network testing environment, software and hardware such as the described in section 5 in order to evaluate the packet processing performance with some specific use cases recommended in (NEMETH *et al.*, 2015):
  - Port Forwarding/L2 forwarding
  - L2 forwarding
  - GRE Encap/Decap

  Evaluating with unidirectional traffic and Small (64B) to large (1518B) packet sizes and with fixed destination and source MAC addresses, IP addresses and ports.

Text Structure

In this Introduction we explained the motivational aspects that justify this work. Also, we give a clear explanation for the objectives of this project.

In Chapter 2 we present a Literature Review. Related toprogramming protocol-independent packet processors (P4) language, OpenDataPlane (ODP) SDK and MACSAD framework architecture are described in the context of our implementation requisites.

In Chapter 3 we take a look at the architecture of the BNG software switch which we explain all the modules an his relationship between. Furthermore, we describe the tables associated to BNG dataplane.

In Chapter 4 we explain the BNG functionalities and features in detail.

In Chapter 5 we show the performance results and compare with related work.

Finally, in Chapter 6 we highlight the results, issues in the implementation process and future work.

2 Literature Review

This section defines three main concepts in our research work: BNG Protocol, another is the P4 language, the third MACSAD compiler and finally an ODP framework description.

Broadband Network Gateway

The first generation network based Broadband Remote Access Server (BRAS) routers was driven for the customer demand by providing High-speed Internet (HSI) to its subscribers, the second generation Ethernet-based Broadband Network Gateway (BNG) routers was driven by subscriber demand of telco services like TV(content broadcast at specific times, e.g., Netflix) delivered in conjunction with voice and HSI services (ALCATEL-LUCENT, 2010).

The today's telco services are provided to its subscribers through an access technology like Digital Subscriber Line (DSL) or Gigabit-capable Passive Optical Network (GPON). The Home Gateway (HG) is the device that brings the interface between the Customer Premise Equipment (CPE) (represents the triple play communications devices like Telephone (Voice), PC (Internet), the Set-top box (TV)) with the BNG and network's core. Such HG functionalities include NAT, local DHCP, GRE sessions, routing, IGMP proxy-routing, etc. This routing device is base equipment for Telco services, enabling Internet access in the home.

One or more HG can be connected to a single DSLAM that sends the traffic to the BNG device that handles the HG traffic to the core IP network, edge routers, and the Internet (See Figure 1).

The network operator provides connectivity, authentication, applications and service network

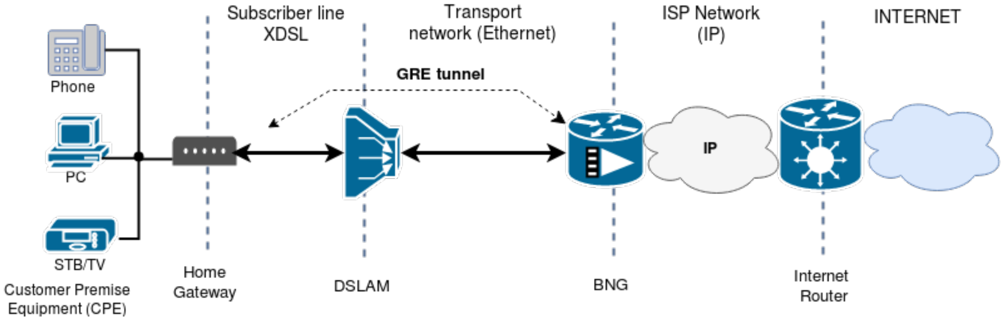


Figure 1 – Access Network Provider Model.

policies to his users, therefore, these procedures involve the premise of session establishment using access communication protocols which are managed in the BNG, the most common protocols to establish session are:

- The PPP over Ethernet (PPPoE): Use the point-to-point (PPP) protocol.
- The IP over Ethernet (IPoE): Use IP protocol that runs between CPE and BNG.
- Generic Routing Encapsulation (GRE V2), encapsulation protocol brings virtual Point-to-Point connections through IP network.

In our implementation, the packet sent or received by the HW are encapsulated with GRE headers, creating a point-to-point link with the BNG, but is just one of the options for packets encapsulation.

Since the BNG centralizes all the functions simplify the management functions like:

- Session management and header cap/decapsulation.
- Interface to Authorization, Authentication and accounting services.
- ARP proxy to manage the requests from the network interface on the BNG side.
- Network Address translation to route the packets towards the operator's core Network.
- Interface to assignment queues and line rate to subscribers.

On the other hand, all these tasks make the structure of network rigid and become difficult to support the protocols and architectures in the current ISP. In (ROBERTO *et al.*, 2013) we found some similar 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 and it composes of netmap and VALE as the packet I/O framework. Our architecture approach goes with the same trend of network function virtualization using Macsad framework that compiles P4 code to bring more flexibility to the data plane and adding support for Hardware Abstraction Layer (HAL).

In the next session, we will describe our architecture design to be rapidly reconfigured using a "programming protocol-independent packet processors" (P4) and MACSAD to generate the datapath Logic codes.

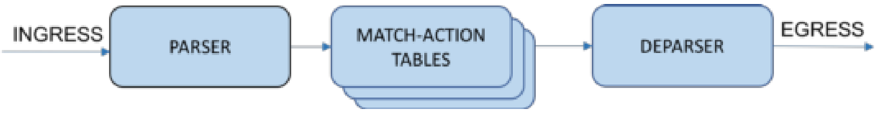


Figure 2 – P4 Abstract Forwarding Model. Source: Adapted from (P *et al.*, 2013).

Programming Protocol-Independent Packet Processors (P4)

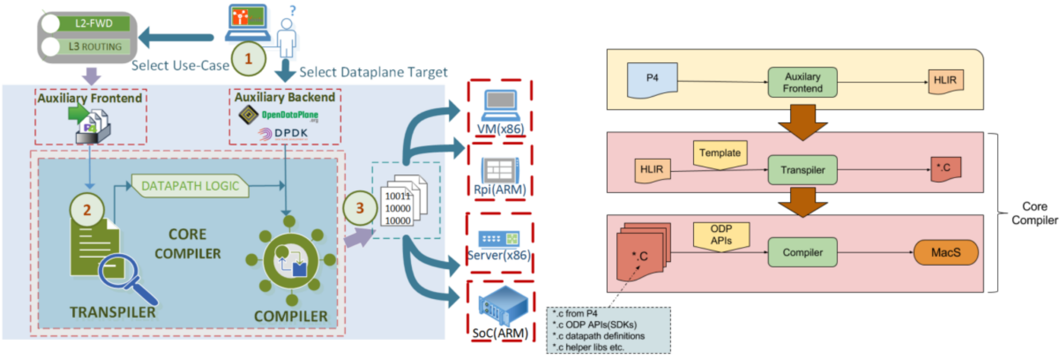
P4 is a high-level language for programming protocol-independent packet processors that define how the pipeline of a network forwarding device should process the packets using the abstract forwarding model (See figure 2). P4 define the header structures and use the parser to extracts the header fields. The pipeline is defined through a series of match-action tables, which execute one or more actions like packet forwarding, drop and so on. This tables can be changed and accessed at "runtime" through a controller software to add, remove and modify table entries and finally, de-parser writes the header fields back before sending the packets to the output port. The three main advantages of P4 are:

1. Reconfigurability in the field: Programmers should be able to change the way how the switch process the packets once that was deployed.
2. Protocol independence: Capability to deploy any protocol in a switch.
3. Target independence: To describe packet-processing functionality independent of the hardware where it has been deployed (P *et al.*, 2013).

Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD)

This tool brings an environment to compile and deploy switch for L2/L3 applications automatically generating the datapath code for heterogeneous targets over 10Gbps network interfaces setup (PATRA *et al.*, 2017).
The MACSAD architecture is designed around the following three modules: (See the figure 2a)

- **Auxiliary Frontend:** The P4 code is the MACSAD input that will be converted to an IR representation using the p4-hlir framework that supports different Domain Specific Language (DSL) to generate a High-Level Intermediate Representation (HLIR) to be used in the Macsad core compiler.
- **Auxiliary Backend:** this module provides a common SDK for the Compiler incorporating



(a) Architecture & Use Case Workflow (b) 3-Tier Compilation Process

Figure 3 – Macsad architecture. Source:(PATRA *et al.*, 2017)

the ODP APIs auto-generating support for different platforms with ODP supporting various control protocols like Switch Abstraction Interface (SAI), OpenFlow (OF), and other useful events for buffer like queueing, scheduling, etc.

- **Core Compiler:** Composed of a Transpiler and a Compiler submodule (see the figure 3b), transforms the IR generated by the frontend into the target imaged in association with the auxiliary backend.

The transpiler takes the HLIR input and auto-generates the Datapath Logic codes. Some critical decision is taken in this sub-module like to optimize the 'Dead Code Elimination' by identifying reachability in a graph, decides the type of look-up mechanism to be used, and size and type of tables to be created by. The Macsad compiler creates a set of libraries in 'C' codes for the desired target (x86, x86+DPDK, ARM-SoC).

The BNG P4 code will be added to the system as an input to create the BNG router. Also, Macsad tool brings the generation of high-level ODP APIs to deliver platform abstraction with high performance and hardware-acceleration options.

Open Data Plane (ODP)

Open Data Plane project provides an application programming environment for data plane applications with high performance and portable across a lot of HW platform. The aim of ODP is separate application design from the functional implementation of that design, this because historically it requires that data plane application must be redesigned on changes in network speed and capacity because the applications need to be very integrated with specialized hardware to achieve acceptable performance levels. ODP relies on the Linux kernel itself; it defines some ODP API set to rapid porting ODP application to any platform (See Figure 4), it can define functions and limits like a number of queues or used cores processor. ODP applications can run in parallel with full Linux user processes that implement control

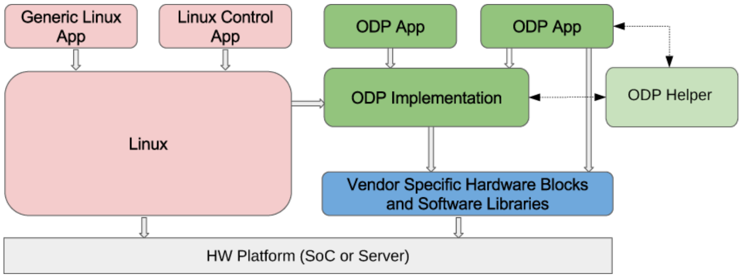

Generic Linux App
Linux Control App
ODP App
ODP App
Linux
ODP Implementation
ODP Helper
Vendor Specific Hardware Blocks and Software Libraries
HW Platform (SoC or Server)

Figure 4 – ODP system architecture. Source:....

and management functions since that these typically do not have critical performance and latency requirements. In that sent, this tool calls the Software Development Kit (SDK) and optimize the features and functionalities for the particular hardware platform (SoC or Server). The ODP APIs is written in the C programing language and are optimized to control related SoC resources such as:

- CPUs (or hardware threads)
- Main memory
- Huge page mappings (how many, what sizes)
- Physical and virtual ports/interfaces
- Packet classification rules
- Scheduler (core groups, algorithms, ordering)
- Hardware queues
- Output traffic management
- hardware Quality of Service (QoS) support.

NFPA

3 Architecture

The Architecture overview is shown in Figure 5. We have main tree processes: (i) Creation of BNG data plane, (ii) Macsad P4_16 support and (iii) Integration with OPD APIs. Finally, this network virtualized device can be used in a commodity server such as executable program.

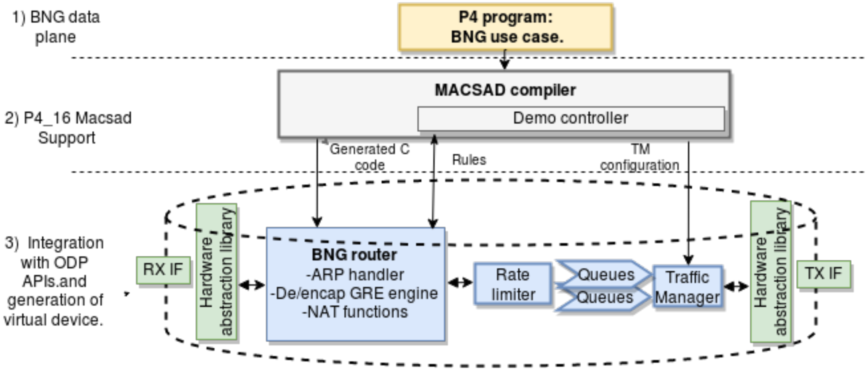


Figure 5 – BNG software architecture.

1. Create a P4 program in order to set the data plane functionalities of the BNG device (BNG data plane).
2. Adds support the the current MACSAD implementation in order to compile the last version of P4 (P4_16).
3. Integration of the BNG compiled and generated version by MACSAD with the ODP APIs, to add others features like Traffic management, Scheduling, and Rate limiter.

Details of processes are explained in the next sections. It is important to highlight that the first process was completed, the second and third are currently in progress and the latter process is included as a scheduled task in Working Plan and Execution Schedule chapter.

Creation of BNG data plane

To support the workload multiples lookup tables are supported either Upload link (UL) and Download Link (DL): a table to store MAC address, a table of CPE based on their IP address routing table, encap/decap GRE, both tables to block any port or IP using a

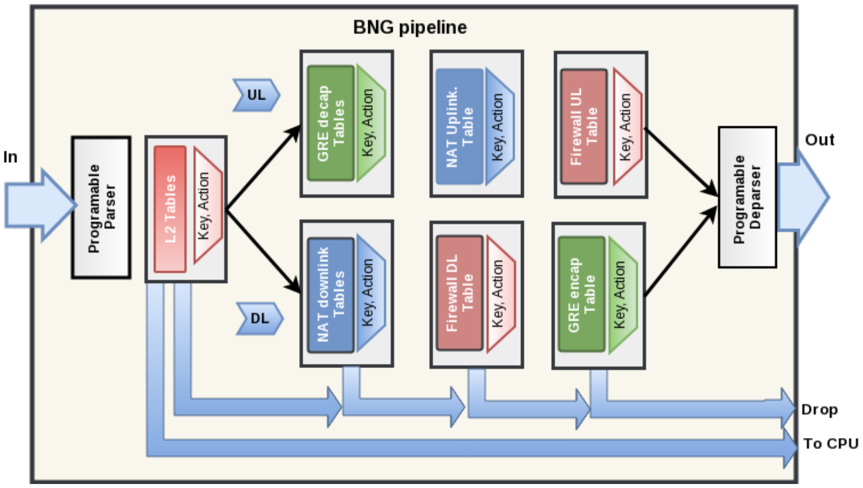


Figure 6 – BNG data plane.

Firewall for UL and DL, both table to Network Address translation (NAT) for UL and DL, the table to limiter the user rate either UL and DL. They are briefly described below.

L2 mac table

When the L2 packet is coming to the BNG switch, it uses MAC learning. Hence, when an ARP packet is received from the CPE, an entry is created (or updated) in a MAC address table swap the source and destination MAC address in the Ethernet header.
Ethernet is the protocol used in the Transport layer. Therefore, the packet header fields are read and rewrite with the new fields in this table in order to forward the packet out by the appropriate port.

Nat table

Basic full-connect NAT for TCP traffic (over IPv4) do both main functions: Translates IPv4 addresses and TCP port numbers of request packets originating from a client on a private network (iAddr: iPort) is mapped to an external address (eAddr: ePort), any packets from iAddr: iPort is sent through eAddr: ePort, the packets that his header fields matching in NAT table are processed in order to rewrite the new header and the packet is forwarded appropriately otherwise when a TCP packet is received on the external interface, for which there is no mapping, the packet is dropped.

GRE tables

The BNG will be able the forward based on the contents of a custom encapsulation header as well as perform normal IP forwarding if the encapsulation header does not exist in the packet. Our BNG is enabled to create point-to-point tunnel mechanism in the intern network to encapsulate the packets with standard GRE packet header structure, as defined by RFC 2784 and RFC 2890 and save the user ID in order to establish the user session. The BNG de-encapsulate the packets originated from CPE to an external address as well as the reverse process when the packet incoming from an external network. The GRE table encap has the outer IP address header field in order add it as new IPv4 header and the GRE header fields in order to enable the new packet.

IPV4 table

The routing table stores the next hop based on the IP address. It's based on the LPM (longest prefix match) implementation. Another table stores information related to the next hop (IP address, port index, etc.). With IPv4 forwarding, the switch must perform the following actions for every packet: (i) update the source and destination MAC addresses, (ii) decrement the time-to-live (TTL) in the IP header, and (iii) forward the packet out the appropriate port. Our BNG device will populate with static rules using a basic controller. Each rule will map an IP address to the MAC address and output port for the next hop.

Parser/Deparser

Our parser will detect the different headers used in the BNG device. It extracts headers from the packet at the current offset into per-packet header instances and marks those instances valid, updating the Parsed Representation of the packet. The parser then indicates as valid byte the correct header and makes a state transition to the next header. The de-parser reverses the process of parsing that emits headers in the proper order. Only headers which are valid are serialized, in our case, the output packet is serialized with the same protocols of the input parser block.

Rate limiter

Some research works [i.e (RODRIGUES *et al.*, 2011),(POPA *et al.*, 2013),(BALLANI *et al.*, 2011)] based on end host-based rate enforcement for fair and guaranteed bandwidth allocation in a multi-tenant datacenter. A datacenter requires a large number of rate limiters, far of the capacity of commodity NICs (e.g., Intel's 82599 10G NIC supports up to 128 rate limiters). Software rate limiters in host network stacks can scale, but this approach implies high CPU overheads hinder support for high-speed links (RADHAKRISHNAN *et al.*, 2014) We implemented a basic Rate limiter in P4 code without slow down performance for every packet to inspect such as defined in RFC 2698 (A Two Rate Three Color Marker - IETF) standard; these describe traffic policing elements like a meter and a dropper. The meter measures the traffic and determines whether or not it exceeds the rate limit, in case that it exceeds the limit the packets could be dropped.

Macsad P4_16 support

Macsad is in charge of converting our BNG P4 program into C code, specifically, it brings support to the version P4_14 (v1.0/v1.1). However, since may of 2017 has been launched the new version P4_16 (v1.0) that compared with the early version doing larges changes regarding syntax and semantics of the language. The new P4_16 specification[1] shows in figure 7 how this has been transformed from a complex language (more than 70 keywords) into a more compact language (less than 40 keywords) accompanied by a library of fundamental constructs that are needed for writing most of the P4 programs.

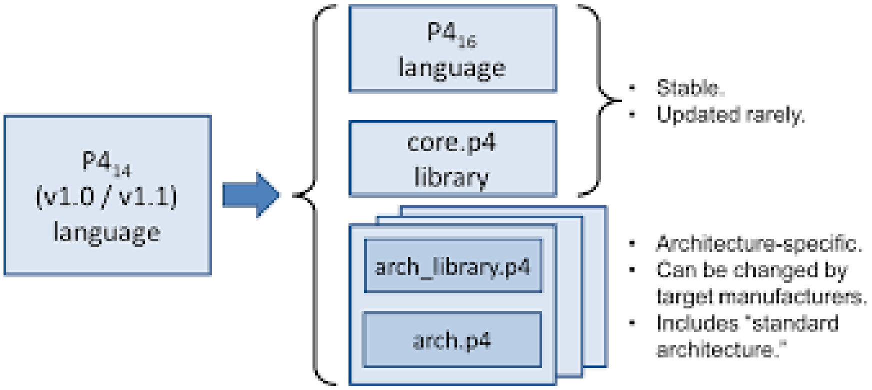

P4₁₄ (v1.0 / v1.1) language
P4₁₆ language
core.p4 library
arch_library.p4
arch.p4
• Stable.
• Updated rarely.
• Architecture-specific.
• Can be changed by target manufacturers.
• Includes "standard architecture."

Figure 7 – Evolution of the language between versions P4_14 (v 1.0 and 1.1) and P4_16.

As we can see the section in the Frontend layer, MACSAD is using p4-hlir project which translates P4 programs into High-Level Intermediate Representation (HLIR) that creates a suitable P4 program representation (See figure 3b), that can be consumed by multiple back-ends, in our case, MACSAD will be used this tool in "Transpiler module", where is access the different P4 top-level objects using these Python ordered dictionary such as p4_parser, p4_tables, p4_actions, p4_headers and so on.

Therefore, our approach for MACSAD update to the new version is using HLIR16 project

[1] https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.pdf

that create a convenient Python representation from a JSON file compiled in P4C compiler from a .p4 source file.

Integration with OPD API's

In order to apply some quality-of-services options for the ISP users, exist some mechanism for achieving these kinds of traffic-management goals in a shared network is through queuing and scheduling. This below features working together, deciding what packets get sent and when; thus scheduling is in charge of sending someone else's packets right now or delaying packets that are arriving too fast. In the following subsection, we will take a look at so-called Weighted fair queuing (WFQ) that provides a straightforward strategy for dividing bandwidth among multiple senders according to preset percentages.

Weighted fair queuing (WFQ)

This is a data packet scheduling algorithm based on both Fair Queueing (FQ) and the generalized processor sharing policy (GPS), where instead of giving each class an equal share, we assign each class a different percentage. For example, If all four input classes A,B,C,D are active, then each gets 25% of the total bandwidth (see figure **??**), just as for a flat four-input-class fair queuing structure. However, when only A, B and C are active, and D is idle. A, B and C each get 33%, hence it lets to allocate bandwidth according to administratively determined percentages.
ODP provides a suite of APIs that control traffic shaping and Quality of Service (QoS). Tx processing using Traffic manager and RX processing involves the use of Scheduler, both processes are described around in the next subsections.

Scheduled RX Processing

Scheduled RX processing is performed via the Scheduler and is requested when a PktIO is opened with an in_mode of ODP_PKTIN_MODE_SCHED. In a basic use, it associates the PktIO input event queues created by odp_pktin_queue_config() with the scheduler. The hash function could be employed to distribute input packets among multiple input queues and instead of these being plain queues they have scheduled queues and have associated scheduling attributes like the priority, scheduler group, and synchronization mode (parallel, atomic, ordered).

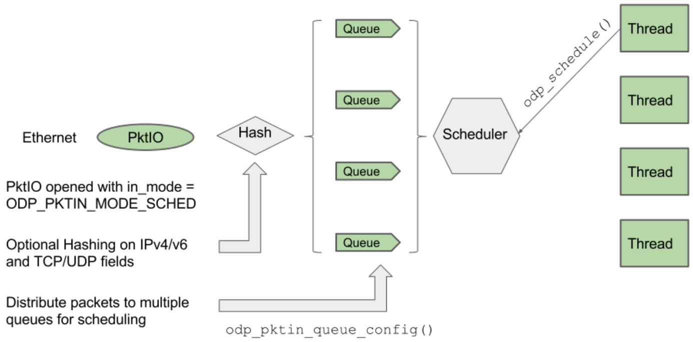

Ethernet
PktIO
Hash
Queue
Queue
Queue
Queue
Scheduler
odp_schedule()
Thread
Thread
Thread
Thread
PktIO opened with in_mode = ODP_PKTIN_MODE_SCHED
Optional Hashing on IPv4/v6 and TCP/UDP fields
Distribute packets to multiple queues for scheduling
odp_pktin_queue_config()

Figure 8 – PktIO SCHED Mode Receive Processing. Source

Inbound packets are classified and put on queues associated with their target class of service which are themselves scheduled to threads (see figure 8).

Scheduled TX Processing

Scheduled TX processing is performed via the ODP Traffic Manager and is requested when a PktIO is opened with an out_mode of ODP_PKTOUT_MODE_TM. The ODP Traffic Manager accepts packets from input queues and applies strict priority scheduling such as weighted fair queueing scheduling and bandwidth controls to decide which input packet should be chosen as the next output packet and when this output packet can be sent onwards. Weighted Fair Queuing (WFQ) is used for priority assign from multiple input packets with the same priority. Each input can be assigned a weight in the range MIN_WFQ_WEIGHT to MAX_WFQ_WEIGHT (nominally 1..255) that affects the way the algorithm chooses the next packet. If all of the weights are equal AND all of the input packets are the same length then the algorithm is equivalent to a round-robin scheduling.

A TM system is composed of Tm_nodes are "entity"/object. It lets interconnect, and interplay of a multi-level "tree" of tm_nodes can allow the user to specify some very sophisticated behaviors. Each tm_node can contain a set of WFQ scheduler (see figure **??**). Each node contains a set of "fan-in" connections to preceding tm_queues or tm_nodes.

4 Development

This chapter covers the software switch development. In section 4.1, we give details about the implementation of the most important BNG features.

Software switch implementation

In this section we discuss how we implemented the BNG software switch adding several changes to the base switch - using C [1], the switch native programming language, and C++ - in order to support all features and keep it as simple as possible.

soporte a primitivas

- add header
- remove header
- drop

[1] In this chapter there will be two common words: struct and structure. While struct is a P4 language keyword and structure is a more generic word for a collection of data variables, both will be used to denote a C struct.

5 Evaluation

In this chapter we evaluate our work in terms of the requisites presented on section 1.1. The first section shows in numbers how many features are covered by the software switch. Subsequently, in the next section, we present the results of performance benchmarks tests.

Experimental Evaluation

We propose two methods to evaluate our prototype: The first one focused in asses the functionality of the pipeline and controller, therefore, this evaluation will be present along BNG development with virtual interfaces, the second one is using our BNG software switch with MACSAD tool in a Linux server to run, compile and assess the performance in a real environment with 10G NIC. Details below:

Functional Evaluation

We assess the functionality of the control plane of the BNG software switch using a packet generator (Scapy) to generate and transmit packets through it and a network protocol analyzer (Wireshark) in a development environment to verify the sent and received packets along the virtual interfaces. The BNG functions to be evaluated are:

- Lookup tables
- L3 forwarding
- Mac learning
- Encap/Decap GRE header
- Network address translation
- Traffic manager
- Controller functions like send entry tables

Performance Evaluation

- It is important to judge the performance of BNG device, considering that we could have millions of packets trough it. In order to validate the BNG software switch in a real network environment, we will compile on the Macsad tool in a commodity server, where the BNG software switch interfaces are connected to the Network Function Performance analyzer (NFPA (**??**)) node via two 10G links to generates test traffic (See figure 9).

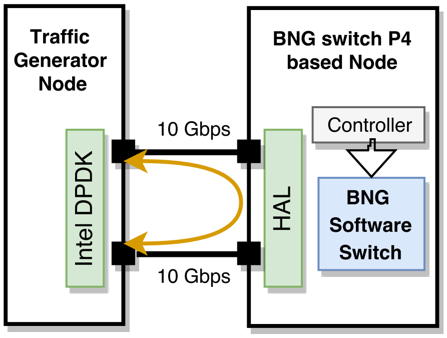


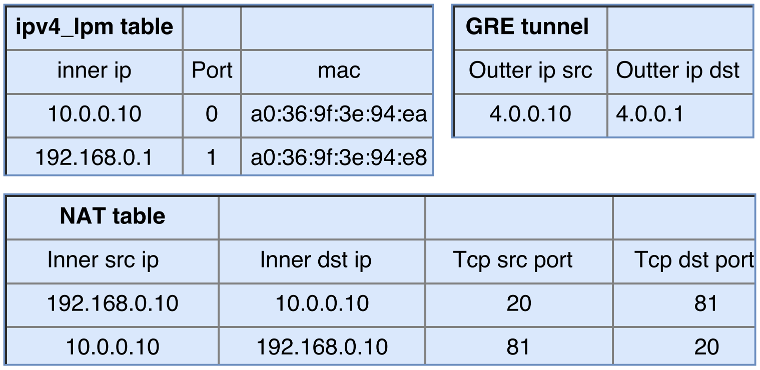
Figure 9 – BNG performance evaluation scenario.

This proof-of-concept implementation, will be run over Macsad server than will receive and forward the packets from Port 0 to Port 1, the NFPA node linked with MACSAD will send the packets and analyze the throughput in packets per second (pps) and bits per second (bps).

For the BNG best-case test, the NFPA node will transmit packets with a fixed source and destination MAC addresses, and IP addresses, TCP port, and tunnel IP addresses, the configurations of the tables are presented, in figure 10 For complex use case, the NFPA provides a wide selection of traffic traces with different packet headers and sizes, for example with 1 to 1 million of packets with different flows and each one having different destination and source MAC addresses, different source and destination IP addresses, different TCP ports, different GRE IP addresses plus different source and destination port.

Feature Completeness

Evaluating the proper operation of the BNG switch features is not a trivial task. For example, testing all flow match fields combinations would require creation of a large number of flows and packets, making manual tests very time consuming. For this reason, automatic



ipv4_lpm table		
inner ip	Port	mac
10.0.0.10	0	a0:36:9f:3e:94:ea
192.168.0.1	1	a0:36:9f:3e:94:e8

GRE tunnel	
Outter ip src	Outter ip dst
4.0.0.10	4.0.0.1

NAT table			
Inner src ip	Inner dst ip	Tcp src port	Tcp dst port
192.168.0.10	10.0.0.10	20	81
10.0.0.10	192.168.0.10	81	20

Figure 10 – Best-case Tables information.

test frameworks, discussed on section **??**, are the best options to test the switch functionality in order to evaluate feature completeness.

Performance Benchmarks

One of the software switch requirements listed on chapter 1 is to reach a maximum throughput of at least 10Gb/s. For this reason we evaluated the switch performance in terms of network metrics. In this section we show how the switch performs for different packet sizes in comparison with different Pktio. Also, we investigated how performance is affected by the number of flows and by the number of tables traversed to match a packet.

The machine configuration used to perform measurement tests are listed in the box below.

- **Processor**: 8x Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz
- **Memory**: 6003MB
- **Operating**: System Ubuntu 16.04 LTS

Maximum Throughput

This test evaluates the maximum forwarding rate the software switch can reach in comparison to other userspace implementations.

The setup for maximum throughput evaluation is the following:

- A running instance of the software switch with two virtual interfaces - Port 1 and Port 2 - attached.
- One flow installed in the Flow Table to match a packet sent to Port 1 with destiny external network... The action is an output packet to Port 2.
- A packet traffic generator. We used a NFPA tool (ISKRATEL, 2003)... .
- A script running in the Linux terminal checking the current packet rate on Port 2.

The test starts by installing the flow in the switch Flow Table. Afterwards, we inject packets, using the traffic generator, directly into Port 1. The bandwidth results of Port 2, reported by the script, are used to calculate the maximum rate. Two transmission measurements were made for 7 packet sizes: for packets of 64Kb, 128kb, 256kb, 512kb, 1024kb, 1280kb and bigger packets of 1500Kb. Figure **??** shows results for both experiments.

Switch forwarding performance for small packets is evaluated in Kilo packets per second (Kpps). Figure **??** shows that ofsoftswitch13 can handle xx.xx Kpps. This result is very far from xxx and approximatelly 32% more efficient than xxx. Bigger packets are measured in Megabits per second. Results presented in Figure **??** show that ...

6 Conclusion

Results

In this section we list positive results achieved on the dissemination of our work:

- **Publications.** This work gave origin to xxx publications. The first paper is about NAT .... The second is an invited paper bringing ...

Future Work

Each architectural component from the software switch has space for improvements. New algorithms and data structures are objects of study for the Flow Table matching. More complex and precise algorithms for rate limiting might be considered for better Meter Table performance. As for groups, new bucket select types may be a subject for academic research.

While there are open ideas for further research and development, some optimizations and features are planned for the software switch in the medium-term. These major improvements are listed below:

Bibliography

ALCATEL-LUCENT. Evolution of the Broadband Network Gateway. 2010. Cited on page 5.

ARORA, D. *Proactive Routing in Scalable Data Centers with PARIS*. 2013. Not cited on the text.

BALLANI, H.; COSTA, P.; KARAGIANNIS, T.; ROWSTRON, A. Towards predictable datacenter networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 41, n. 4, p. 242–253, ago. 2011. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2043164.2018465>. Cited on page 13.

BIANCHI, G.; BONOLA, M.; CAPONE, A.; CASCONE, C. Openstate: Programming platform-independent stateful openflow applications inside the switch. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 44, n. 2, p. 44–51, abr. 2014. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2602204.2602211>. Not cited on the text.

BRADEN, R.; BORMAN, D.; PARTRIDGE, C. *Computing the Internet checksum*. IETF, 1988. RFC 1071. (Request for Comments, 1071). Updated by RFC 1141. Disponível em: <http://www.ietf.org/rfc/rfc1071.txt>. Not cited on the text.

DEERING, S.; HINDEN, R. *Internet Protocol, Version 6 (IPv6) Specification*. IETF, 1998. RFC 2460 (Draft Standard). (Request for Comments, 2460). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Disponível em: <http://www.ietf.org/rfc/rfc2460.txt>. Not cited on the text.

GIT. *git –distributed-even-if-your-workflow-isnt*. 2005. <http://git-scm.com/>. [accessed: 23-Fev-2015]. Not cited on the text.

HAN, B.; GOPALAKRISHNAN, V.; JI, L.; LEE, S. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, v. 53, n. 2, p. 90–97, Feb 2015. ISSN 0163-6804. Cited on page 1.

ISKRATEL, M. J. *packeth - Ethernet packet generator*. 2003. [accessed: 01-Mar-2015]. Disponível em: <http://packeth.sourceforge.net/packeth/Home.html>. Cited on page 22.

NAGAPPAN, N.; MAXIMILIEN, E. M.; BHAT, T.; WILLIAMS, L. Realizing quality improvement through test driven development: Results and experiences of four industrial teams. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 13, n. 3, p. 289–302, jun. 2008. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9062-z>. Not cited on the text.

NBEE. *NetBee Library*. 2012. <http://www.nbee.org/>. [accessed: 11-Nov-2014]. Not cited on the text.

NEMETH, B.; SIMONART, X.; OLIVER, N.; LAMOTTE, W. The limits of architectural abstraction in network function virtualization. p. 633–639, May 2015. ISSN 1573-0077. Cited on page 2.

ONOS. *ONOS Segment Routing*. 2014. <https://wiki.onosproject.org/display/ONOS/Installation+Guide>. [accessed: 03-Mar-2015]. Not cited on the text.

OpenWRT. 2004. Disponível em: <http://www.openwrt.org>. Not cited on the text.

P, B.; D, D.; M, I.; MCKEOWN, N.; REXFORD, J.; SCHLESINGER, C.; TALAYCO, D.; VAHDAT, A.; VARGHESE, G.; W, D. Programming Protocol-Independent Packet Processors. 2013. ISSN 01464833. Disponível em: <http://arxiv.org/abs/1312.1719>. Cited 2 times on page(s) xvii and 7.

PATRA, P.; ROTHENBERG, C.; PONGRACZ, G. MACSAD: High performance dataplane applications on the move. *IEEE International Conference on High Performance Switching and Routing, HPSR*, v. 2017-June, 2017. ISSN 23255609. Cited 3 times on page(s) xvii, 7, and 8.

POPA, L.; YALAGANDULA, P.; BANERJEE, S.; MOGUL, J.; TURNER, Y.; SANTOS, J. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. v. 43, p. 351–362, 08 2013. Cited on page 13.

RADHAKRISHNAN, S.; GENG, Y.; JEYAKUMAR, V.; KABBANI, A.; PORTER, G.; VAHDAT, A. Senic: Scalable nic for end-host rate limiting. USENIX Association, Berkeley, CA, USA, p. 475–488, 2014. Disponível em: <http://dl.acm.org/citation.cfm?id=2616448.2616492>. Cited on page 14.

REITBLATT, M.; CANINI, M.; GUHA, A.; FOSTER, N. Fattire: Declarative fault tolerance for software-defined networks. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 109–114. ISBN 978-1-4503-2178-5. Disponível em: <http://doi.acm.org/10.1145/2491185.2491187>. Not cited on the text.

RISSO, F.; BALDI, M. Netpdl: An extensible xml-based language for packet header description. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 50, n. 5, p. 688–706, abr. 2006. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2005.05.029>. Not cited on the text.

ROBERTO, B.; THOMAS, D.; H, F.; A, M.; M, J.; N, S.; K, H.-J. Rethinking Access Networks with High Performance Virtual Software BRASes. *EWSDN*, 2013. Cited on page 6.

RODRIGUES, H.; SANTOS, J. R.; TURNER, Y.; SOARES, P.; GUEDES, D. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. USENIX Association, Berkeley, CA, USA, p. 6–6, 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=2001555.2001561>. Cited on page 13.

RUPARELIA, N. B. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 35, n. 3, p. 8–13, maio 2010. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/1764810.1764814>. Not cited on the text.

SHOURMASTI, K. S. *Stochastic Switching Using OpenFlow*. [S.l.]: Institutt for telematikk, 2013. 70 p. Not cited on the text.

STONE, J.; STEWART, R.; OTIS, D. *Stream Control Transmission Protocol (SCTP) Checksum Change*. IETF, 2002. RFC 3309 (Proposed Standard). (Request for Comments, 3309). Obsoleted by RFC 4960. Disponível em: <http://www.ietf.org/rfc/rfc3309.txt>. Not cited on the text.

TANENBAUM, A. Computer networks. In: _____. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. p. 401. ISBN 0130661023. Not cited on the text.

The Organization for Economic Co-operation and Development (OECD). Improving Networks and Services Through Convergence discussion paper. *2016 Ministerial Meetign*, 2016. Cited on page 1.

TOURRILHES, J. *OpenFlow prototyping*. 2013. <https://github.com/jean2/ofsoftswitch13>. [accessed: 11-Nov-2014]. Not cited on the text.

YIAKOUMIS, Y.; SCHULZ-ZANDER, J.; ZHU, J. *Pantou : OpenFlow 1.0 for OpenWRT*. 2011. Disponível em: <http://www.openflow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT>. Not cited on the text.

Annex

ANNEX A – BNG P4 code

ANNEX B – Publications

Three papers were published during this work and are listed below.

- Juan Sebastian Mejia, Daniel Feferman Christian Esteve Rothenberg. "NAT....". In Salão de Ferramentas XXXII Simpósio Brasileiro de Redes de Computadores - CSBS'2018, Florianópolis, 5 a 9 de Maio de 2014.