Received: 14 February 2020 | Revised: 15 June 2020 | Accepted: 13 July 2020

DOI: 10.1002/nem.2134

SPECIAL ISSUE PAPER

WILEY

OpenBNG: Central office network functions on programmable data plane hardware

Ralf Kundel[1] | Leonhard Nobach[2] | Jeremias Blendin[3] | Wilfried Maas[2] | Andreas Zimber[2] | Hans-Joerg Kolbe[2] | Georg Schyguda[2] | Vladimir Gurevich[3] | Rhaban Hark[1] | Boris Koldehofe[1,4] | Ralf Steinmetz[1]

[1]Multimedia Communications Lab, Technical University of Darmstadt, Darmstadt, Germany

[2]Fixed Mobile Engineering Deutschland, Deutsche Telekom Technik GmbH, Darmstadt, Germany

[3]Barefoot Networks, an Intel Company, Santa Clara, California, USA

[4]Faculty of Science and Engineering, University of Groningen, Groningen, The Netherlands

Correspondence
Ralf Kundel, Multimedia Communications Lab, Technical University of Darmstadt, Darmstadt, Germany.
Email: ralf.kundel@kom.tu-darmstadt.de

Funding information
Deutsche Forschungsgemeinschaft, Grant/Award Number: Collaborative Research Center (CRC) 1053 - MAKI; German Research Foundation (DFG)

Summary

Telecommunication providers continuously evolve their network infrastructure by increasing performance, lowering time to market, providing new services, and reducing the cost of the infrastructure and its operation. Network function virtualization (NFV) on commodity hardware offers an attractive, low-cost platform to establish innovations much faster than with purpose-built hardware products. Unfortunately, implementing NFV on commodity processors does not match the performance requirements of the high-throughput data plane components in large carrier access networks. Therefore, programmable hardware architectures like field programmable gate arrays (FPGAs), network processors, and switch silicon supporting the flexibility of the P4 language offer a promising way to account for both performance requirements and the demand to quickly introduce innovations into networks. In this article, we propose a way to offer residential network access with programmable packet processing architectures. On the basis of the highly flexible P4 programming language, we present a design and open source implementation of a broadband network gateway (BNG) data plane that meets the challenging demands of BNGs in carrier-grade environments. In addition, we introduce a concept of hybrid openBNG design, realizing the required hierarchical quality of service (HQoS) functionality in a subsequent FPGA. The proposed evaluation results show the desired performance characteristics, and our proposed design together with upcoming P4 hardware can offer a giant leap towards highest performance NFV network access.

1 | INTRODUCTION

Internet service providers (ISPs) are challenged by competition, regulators, and content providers to deliver high-performance services to residential subscribers at ever lower costs. Essential cost drivers are the high-performance network functions required at the access edge. From a functional perspective, besides offering Internet connectivity, the access edge has to provide authorization and implement the quality properties of Internet access contracts of residential subscribers. Traditionally, residential network access is provided by physical network functions called *broadband network gateways (BNGs)*, which are purpose-built devices that provide high performance, but tend to be costly. One approach to reduce the costs for network operators is *network function virtualization (NFV).* NFV aims to increase cost-efficiency and flexibility by implementing network functions on cost-efficient commodity servers, commonly based on the `x86` or `ARM` processing architecture. One way to realize NFV in ISP networks is *Central Office Re-architected as a Datacenter (CORD)*,[1] which aims to transfer ideas from the data center world to ISP networks. However, for high-performance packet forwarding, softwarized approaches have unfavorable performance characteristics compared to fixed hardware solutions.

Our previous work investigated how commodity *bare-metal switches* could be adapted to provide the function of residential network access.[2] Commodity switch silicon in 2016 was sufficient to implement the essential properties of a small-scale BNG with a very high performance, but not with all features needed by a large-scale network operator, like *Point-to-Point Protocol over Ethernet (PPPoE)*encapsulation/decapsulation or the required scalability.

With newer hardware designs and the highly flexible P4 pipeline description language,[3] packet processing can be adapted to a variety of use cases. P4 shows the potential to fulfill all functional requirements that are desired for a BNG data plane. Furthermore, it is expected that numerous vendors of packet processing hardware will support the P4 programming language to configure their upcoming switching chips (ASICs) or network processors. ASICs or network processors generally provide much better performance than software-based NFV approaches for data plane tasks. The goal of this article, which is an extended version the previously published paper "P4-BNG: Central office network functions on programmable packet pipelines,"[4] is to present a P4-based design and implementation of a BNG data plane and a field programmable gate array (FPGA)-based queueing chip, which runs in a CORD environment and fulfills all requirements of a large-scale telecommunication provider. In detail, the contributions of our work are as follows:

- We analyze the functional requirements, including QoS and queueing needs, of a large-scale telecommunications provider's BNG in Section 3.
- We present a design and open source implementation a BNG data plane network function in a CORD-based design for P4 targets, fulfilling these requirements.
- We introduce a disaggregated architecture of P4 data plane and QoS-FPGA enabling a flexible and powerful queueing system in Section 5.
- We provide an abstraction layer between a generic BNG and the runtime environment.
- We describe functional tests to evaluate the proposed packet processing pipeline and present evaluation results of a first prototype in Section 7.

The outline of this article is structured as follow: After providing an overview over state of the art and related approaches, we present the results of the requirements analysis in Section 3. In Sections 4 and 5, we present the design of this work. Finally, we discuss the implementation and present some evaluation results.

2 | BACKGROUND AND RELATED WORK

To benefit from fast, programmable packet processing hardware, our implementation of the BNG data plane is using the P4 pipeline description language and targets upcoming programmable hardware devices. Our implementation is designed as a part of the CORD architecture which supports the execution of central-office functionality, using VNFs on commodity hardware. In the following, we provide an overview of the CORD project, the P4 programming language, and related work.

2.1 | Broadband network gateways

Central offices, the points connecting subscribers to the telecommunications infrastructure, have evolved in the last 100 years from analog, human-driven switching centers to digital packet gateways. Today, residential network access functions in the central office are integrated in the BNG, mostly implemented on purpose-built hardware. The BNG function has been specified by several technical reports of the Broadband Forum (TR-101,[5] TR-145,[6] and TR-178[7]). According to IETF internet drafts, a BNG is defined as a server which routes traffic to and from broadband access devices, e.g., DSLAMs, on an ISP network.[8] Further terms are *Network Access Server (NAS)*[9] and *Broadband Remote Access Server (BRAS)*.[10] In this work, we use the common term *BNG*.

2.2 | SDN for telecommunication networks

A major contribution of *software-defined networking (SDN)* architectures is the separation of the control plane and the data plane.[11] In SDN, the control plane is commonly executed on standard hardware and is intended for signaling traffic only with minor overall performance requirements, whereas the data plane is still realized on conventional ASICs. With the flexibility of standard processors, the control plane can, however, solve complex tasks.

The usability of SDN for telecommunications providers has been investigated within the project SPARC.[12] The focus of the latter research project was the disaggregation of the data plane and multiple, hierarchical arranged, control planes. However, the project found the existing OpenFlow[13] protocol insufficient for the task and, therefore, proposed numerous improvements to it.

Recently, the *Open Networking Foundation (ONF)* conceived the CORD project[1] to overcome the dependency on vendors and proprietary hardware in carrier networks. Based on a combination of capabilities provided by SDN and NFV,[14] the CORD project opens up new possibilities to bring "economies of scale" and agility of data centers into central offices. With CORD, network functionality of central offices is implemented as VNFs, based on open-source software components, such as Docker,[15] OpenStack,[16] and ONOS.[17] According to the authors, the economies of scale of commodity hardware and software promises reduced costs and vendor lock-ins.

CORD focuses on *Gigabit Passive Optical Networks (GPON)* as the access technology only which is not realistic in large-scale real-world deployments which typically consist of several technologies. Nevertheless, other technologies like XGS-PON, G.fast,[18] or VDSL fit into the system model as well. The distinction of the access lines is ensured through VLAN tags, which are added or removed by the access node (AN); in the concrete case of GPON, this is an Optical Line Termination (OLT), shown in Figure 1. For each subscriber, a dedicated VNF instance, a virtual subscriber gateway, provides traffic processing. If a new subscriber establishes a connection to the CORD POD, a new virtual subscriber gateway is started, and the routes within the POD are established. From now on, all associated incoming packets are forwarded to the subscriber's virtual gateway, which does the header processing and forwards the packets between the core and the subscriber. The end-to-end service chain consists of three different parts, a virtual OLT, a virtual service gateway, and a virtual router. CORD claims that up to 1000 subscribers per server can be handled and a round-trip latency through the POD below 1 ms is possible. Although this very promising approach achieves flexibility and vendor independence, the performance of the virtual subscriber gateway Docker containers suffers from the software implementation of the data path, compared to using a hardware implementation.[19] The CORD project can be divided into three subparts for residential (R-CORD), mobile (M-CORD), and enterprise (E-CORD) access.

2.3 | OpenFlow

The fundamental idea of SDN is the disaggregation and by that separation of control plane and data plane. For that, an open and specified interface for information exchange between these planes is needed. The OpenFlow protocol, introduced in 2008,[13] became a de facto standard for this control interface, at least for traditional SDN. Implementing complex network functions with commodity OpenFlow hardware is not possible for numerous reasons. First, the OpenFlow protocol does not support the use of special protocols such as PPPoE.[2] Second, not all functionalities that are available in the OpenFlow protocol are available in actual OpenFlow switches. Thus, even a newer version of the protocol, e.g., supporting PPPoE, would be constraint by hardware limitations of switching hardware.

2.4 | P4 and reconfigurable hardware

The majority of currently available devices, including OpenFlow-enabled switches, have fixed functionality that can only be modified within narrow limits. Furthermore, the design and production process for new ASICs is very

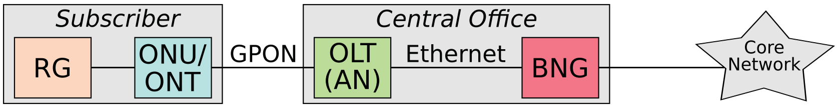


FIGURE 1 Homogeneous access topology assumption of the CORD project[1] with only GPON as access technology

time- and cost-intensive. The flexibility of software for VNFs is branded by limited bandwidth (PCIe, NIC, CPU) and its higher latency. The P4 language,[3] introduced in 2014, is designed to describe the pipeline behavior of reconfigurable network devices and combines the benefits of flexible software and high-performance hardware. This language has four main goals: (1) reconfiguration of deployed hardware, (2) independence of any network protocol, (3) independence of the specific target hardware, and (4) the ability to generate control plane APIs for a given data plane program. P4 can be used to describe any network protocol headers and processing algorithms which can be expressed by a Direct Acyclic Graph (DAG). Thereby, new protocols can be added to the target pipeline without buying new hardware—only by updating the configuration. In 2016, an enhanced version of P4, called $P4_{16}$,[20] has been published, a more general, structured, and feature-enriched version. For example, the description of a pipeline which does not fit on the $P4_{14}$ reference Pipeline, e.g., FPGA based P4 switches, can be described with $P4_{16}$ as well.

Different vendors started introducing programmable hardware for packet processing in the last years, for example, *Barefoot Networks* and *Netronome*. Furthermore, FPGAs can be used to implement pipelines described in P4 or similar languages, using high-level synthesis tools.[21] In this work, we show how this language can be used to describe a high-performance BNG data plane.

2.5 | Programmable hardware for building VNFs

As already found by related work, implementing a high-performance VNF *offloading* to hardware accelerators should be considered for two main reasons. First, typically the performance, e.g., packets per seconds and latency, is much better. Second, a much better efficiency factor reduces the energy costs and required number of devices strongly[22] which is the case for many computing applications and not only for network functionality. The authors of previous approaches for hardware accelerated VNF execution[23,24] assume, as the traditional VNF approach,[25] a “off the shelf” server hardware which is extended by hardware accelerators like FPGAs or GPUs.

However, the use of hardware accelerators does not necessarily implies an “off the shelf” server centric system model as some of them can be used as stand-alone devices. One very promising hardware acceleration technology is FPGA. Any digital circuit, which includes the behavioral model of network switching ASICs, can be realized on an FPGA. The typically performance is somewhere in between of a commodity x86 server and an ASIC with fixed functionality. However, compared to fixed ASICs, the flexibility regarding digital circuit adaption and development costs is much better. The P4-NetFPGA project[26] provides a high-level work flow for configuring FPGAs as network processors with a high-level programming language. Many works build upon this work flow framework, e.g., Xiong and Zilberman[27] have shown how to integrate a complex traffic classification framework into FPGA-based P4 data planes. Even though FPGAs can be used as stand-alone devices, PCIe is still used as control plane interface. The data plane I/O is typically realized on the FPGA itself.

Also, in other areas of communications, the usage of P4-based FPGA data planes becomes more convenient. For example, Ricart-Sanchez et al.[28] realized a 5G data plane for edge-to-core communication on FPGAs. Similar to this work, Singh et al.[29] realized the data plane of a virtual Evolved Packet Gateway (vEPG) in 4G access networks on a P4 programmable switch. Their evaluation results have shown a low latency of less than $2\,\mu$ s and claim a low jitter which underlines our observations. Besides realizing conventional data plane functionality in programmable data planes, Shah et al. realized 4G control plane message handling in the data plane.[30] They observed a huge number of 4G signaling messages (S1 release + service request) which can be processed directly within the P4 programmable data plane in order to accelerate the message processing and unload the software-based control plane.

2.6 | Programmable queues for flexible packet scheduling

The need of flexible queue programming has been discussed in related work before. In the context of advanced Active Queue Management (AQM) algorithms, Sivaraman et al. have already claimed that there is not one single optimal queueing mechanism. Therefore, they proposed the idea of FPGAs for scheduling and AQM functionality.[31] As FPGAs are not suitable for highest throughput applications such as >1 Tbit/s switches, other approaches for programming queues are proposed as well. Three years later, the same authors introduced Push-In First-out (PIFO) queues as a building block for programmable queues and schedulers.[32] This approach allows, compared to current fixed FIFO queue approaches, a much higher flexibility. Compared to fully flexible FPGAs, this approach can technically provide higher packet rates but still with limited flexibility. However, a realization of PIFO queues within switching ASICs seems to be quite challenging.

The work of Alcoz et al. built upon the PIFO queue approach and the challenging realization in hardware. They proposed the idea of approximating a single PIFO queue by a set of multiple FIFO queues with ascending priorities.[33] In this way, a very similar behavior can be realized, but the implementation in switching ASICs is less challenging, and the authors have shown the feasibility by a prototypical implementation based upon existing P4 programmable chips. The construct of (SP-)PIFO queues would fit well into the requirements of a hierarchical quality of service (HQoS) system for BNGs as multiple priorities and a shared traffic shaper could be realized by that.

Last, we would like to focus on the work of Sharma et al.[34] They proposed the idea of calendar queues for high-speed packet scheduling. Even though existing programmable data plane ASICs do not support all needed functionalities, the additional features could be added without significant design changes. By having time-constrained priorities per queue, many advanced queueing policies can be realized. We believe that also real-time applications, most mentionable time-sensitive networking (TSN), could benefit from that.

3 | REQUIREMENTS ANALYSIS FOR A CARRIER-GRADE BNG

Implementing a fully fledged P4-BNG requires detailed, in-depth knowledge of all functional requirements for BNGs in a DSL/GPON carrier environment. This section gives an insight into these functional requirements, which we have obtained by an requirements survey, based on the previous studies[35-37] and own requirements.

3.1 | Network access lines and nodes

Access lines are a fundamental part of the BNG model. They are typically the limiting factor for the bandwidth that can be offered to a subscriber. Therefore, a subscriber is effectively sold a bundle of an access line and an Internet access service, established and enforced by a PPPoE session. An AN terminates the access line of a subscriber and provides corresponding virtual wires that connect the residential gateway to the BNG. Virtual wires are implemented using per-access line VLAN tags unique to each AN, which, except for this specific task, does not perform major subscriber-specific functions.

Figure 3 depicts residential Internet access with different access technologies as Fiber-to-the-Home (FTTH) or Fiber-to-the-Curb (FTTC). Dependent on the disposed technology, the AN might be a Optical Line Terminal in case of FTTH or a Multi Service AN (MSAN) for FTTC. As described later in Section 3.3 in detail, all nodes in the access network can be oversubscribed which is typically the case.

3.2 | Functional BNG requirements

A BNG is a system composed of one or multiple devices, which jointly implement residential and business access services. Based on specific configuration parameters, this system assigns network resources including policy restrictions, usually defined by contractual constraints.

In general, the *access service provisioning* process is determined according to four general phases: (1) user discovery and authentication, (2) parameters assignment, (3) access control and features enforcement, and (4) connection monitoring,[37] depicted in Figure 2.

Starting from the top and moving down, the table illustrates different functional services that the BNG system must provide, leading to several tasks of the BNG implementation:

3.2.1 | Tunneling

The network access line, introduced in Section 3.1, describes the virtual connection between a residential gateway and an AN. However, after successful subscriber authentication and authorization, a virtual connection between residential gateway and BNG will be established upon this access line. This connection is typically realized by a tunneling protocol

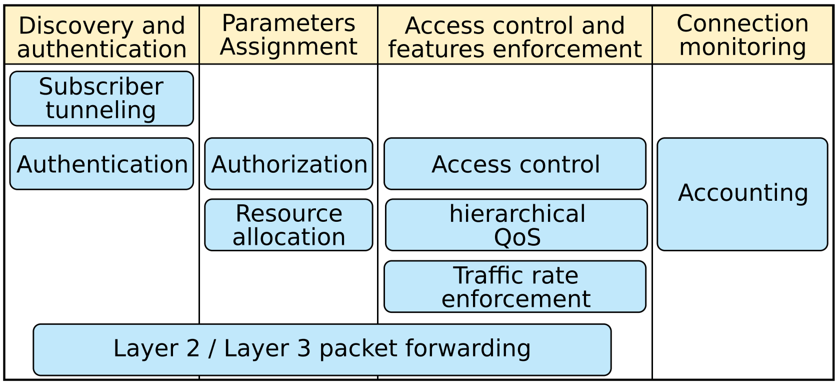


FIGURE 2 Access service creation phases of a BNG system

as L2TP or PPPoE *(RFC 2516)* which we will consider in the following. For each access line, a single PPPoE session is used to authorize the subscriber and enforce restrictions as described in the subscribers' contract. The discovery and session setup phase are conducted by the control plane. For that, the data plane forwards all control packets from the subscriber to the control plane and backwards. After the session setup, the data plane handles PPPoE encapsulation and decapsulation without control plane interaction. Per subscriber line, only a single PPPoE session is required.

3.2.2 | Number of active subscribers per BNG

Due to the economies of scale, many subscribers should be treated by a BNG. According to an Intel blog post,[38] the current number of subscribers per central office is around 5000. However, in the future up to 35 000 can be expected. This implies high demands on a BNG system as all functionality must be provided for thousands of subscribers in parallel.

3.2.3 | IP address assignment to subscribers and reverse path forwarding

Intellectual property (IP) addresses must be assigned to new subscribers from the address pool by the control plane and installed in the data plane. The data plane has to ensure that packets to subscribers are forwarded using the subscriber specific PPPoE tunnel information. Furthermore, packets sent from subscribers have to be filtered by the IP addresses assigned to the subscriber, known as *reverse path forwarding (RFC 3704).*

3.2.4 | QoS control

QoS is in use to separate traffic of different precedence and acuteness, e.g., normal Internet traffic, assured forwarding for services like IPTV, and expedited forwarding for real-time services like voice over IP (VoIP). The corresponding QoS header fields of packets, ingressing from subscriber side at the BNG, cannot be trusted. This is why they should be overwritten by the BNG in order to prevent abuse of traffic classes by identifying QoS classes based on L3 addresses for traffic from and to subscribers.

Table 1 depicts an example set of QoS classes which are typically present in ISP access networks. QoS awareness should be considered together with traffic shaping policies, further discussed in Section 3.3 as they strongly depend each other.

3.2.5 | Time to live

The time to live (TTL) fields of forwarded packets are reduced in the forwarding process. If the TTL reaches zero, the packet must be dropped, and an ICMP notification has to be forwarded to the sender. Since this only happens very seldom, the processing can be done by the control plane.

3.2.6 | MTU and fragmentation

Adding the PPPoE header stack to traffic from the Internet to subscribers requires checking the packet size for violations of the maximum transmission unit (MTU). The residential gateway MTU is expected to be 1500 bytes for IP

TABLE 1 Example ISP QoS classes with ascending priorities

Class	Description
BE$_{\text{low}}$ (0)	The traffic class *Best Effort low priority* (BE$_{\text{low}}$) is used for hotspot services where the residential gateway is a public WIFI access point at the same time. As the customer, paying for this residential Internet access, should not be restricted by third person traffic, an additional traffic class is needed.
BE (1)	*Best Effort (BE)* includes all data streams which belong neither to a priority traffic class nor to BE$_{\text{low}}$. Compared to the other traffic classes, the bandwidth utilization by this class is typically much higher. All "normal traffic" of residential subscriber lines belongs to this class.
LD/LL (2/3)	Low Delay (LD) and Low Loss (LL) are used for enterprise applications of business customers only. For residential access, they are typically not used.
MC (4)	Multicast (MC) is used for television products of ISPs. Duplication is realized at the BNG, and by that, data transfer in the core network is strongly reduced. Note: This includes only live TV offered by the ISP, not (third-party) on-demand video streaming.
VoIP (5)	Many ISPs offer voice over IP (VoIP) services to their customers as well, which require a very low latency and loss rates. Note: All application layer VoIP approaches, e.g., Skype or WhatsApp, are handled as BE traffic.
Ctrl (6)	This traffic class is only used for controlling the ISP network including access network and RGs.

Note: Implementations of different ISPs will vary slightly in the number and type of traffic classes; however, the general structure will persist.

packets, the standard for Ethernet links. Packets transmitted on this link include additional PPPoE and VLAN headers as depicted in Figure 5. While the VLAN header is part of the Ethernet header, the 8 bytes for the PPPoE header decrease the MTU to 1492 bytes (RFC 4638). In case of larger packets, the BNG has two options: (1) fragment and forward or (2) discard the packet and signal the event to the sender by an ICMP message. In case of IPv6 or IPv4 "don't fragment" packets fragmentation is not possible and therefore Path Maximum Transmission Unit discovery (PMTU) or TCP Maximum Segment Size (MSS) determination is usually conducted (RFC 1191). The MSS option of TCP can be influenced by the BNG.

3.2.7 | Multicast

IP multicast is useful for IPTV products of the ISP in order to reduce the total traffic by duplicating packets at the BNG. The multicast traffic can be encapsulated in the same way as other subscriber traffic on the access line. Establishing the multicast streams from upstream sources as well as processing IGMP messages from subscribers is done by the control plane. Similar to PPPoE session setup and control packets, the data plane detects these packets and forwards them to the control plane. The task of the data plane is to handle the packet duplication at the required scale.

3.3 | QoS aware traffic shaping and rate limiting

Rate limiting, shaping and QoS prioritization can be divided into two parts: (1) downstream traffic and (2) upstream traffic. These two parts can be considered independently and will be discussed in the following:

3.3.1 | Downstream QoS

The downstream traffic is queued and shaped at the BNG to ensure that the access network is not overloaded and a constant bandwidth can be guaranteed, based on customer contract and access network limitations. *Note:* Alternatively, a shaping can be performed later in the ANs, if supported by the deployed hardware. As part of our case study, the following requirements could be derived:

Queueing hierarchy

Multiple hierarchical queueing constraints as shown in Figure 3 must be implemented. This is needed as after through passing the BNG, no packet loss must be guaranteed in the access network and multiple nodes are oversubscribed. By

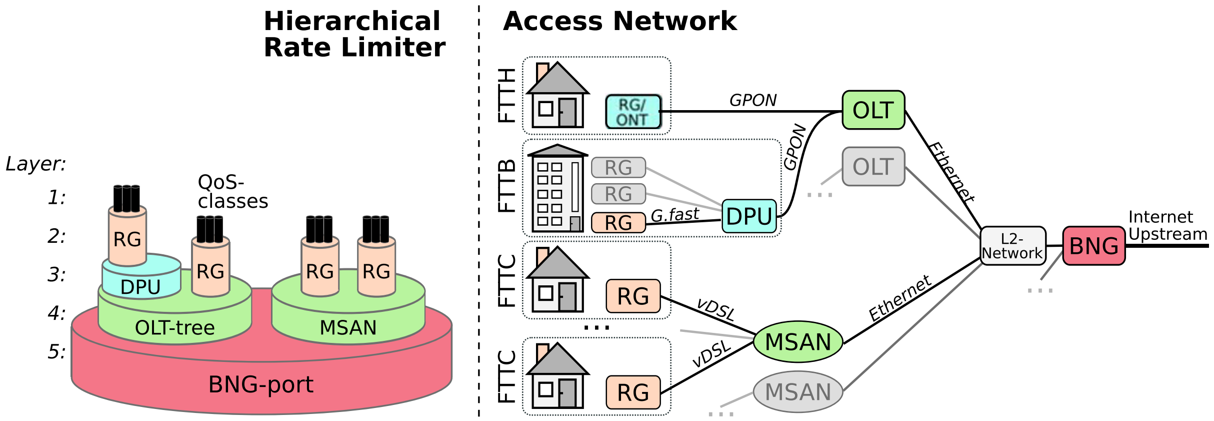


FIGURE 3 Hierarchical access network with nonuniform access technologies and hierarchical rate limiter dependencies. Each node on each layer in the access network tree (visualized on the right side) can be oversubscribed. This implies a hierarchical rate limiter architecture as visualized on the left side representing the bandwidth limit of all access network nodes

that, dependent on the ISP access network structure and further optional requirements, three to six hierarchical and QoS-aware scheduler layers are needed, named HQoS.

Traffic classes

As already introduced in Section 3.2.4 and shown in Table 1, there can be four to eight traffic classes with ascending priorities, latency, and loss requirements. Consequently, each traffic class is modeled by a separate queue. This results in up to eight FIFO queues for each subscriber. Realizing multiple priorities within one queue and QoS-dependent loss rates would be possible as well. However, all packets would suffer under the same queueing delay which causes additional latency. A scheduling decision must be aware of this priority within all queues of a subscriber and between subscribers.

Number of queues

Assuming 5000–35 000 subscribers and four to eight queues per subscriber, a total number between 20 000 and 280 000 FIFO queues are needed. As subscribers must be isolated to each other, sharing queues between subscribers is not a wise idea. Shared queues would imply that *Subscriber A* could suffer under a bloated queue, and by that higher latency, caused by *Subscriber B (C, D, and so on).* Furthermore, bandwidth distribution unfairness can occur as different L4 congestion control mechanism behave differently in a competitive way.[39]

Bandwidth

There are different ways to estimate the total maximum downstream bandwidth of a BNG and by that of the queueing system. Assume each customer has a downlink speed of 100 Mbit/s and there are 10 000 subscribers per BNG; the total maximum downstream bandwidth could be 10 Tbit/s. A more precise upper bound is given by the ingress ports of the BNG. Dependent on the total number of subscribers, BNG generation, and carrier requirements, a total downstream bandwidth between 20 and 400 Gbit/s is realistic.

Memory

The total required memory can be computed by the maximum buffering time and the bandwidth. With a buffer of 50 ms and 200 Gbit/s, the memory must be larger than 1.25 GB in order to avoid overflow. However, these numbers are an upper bound as it is not realistic that all queues, e.g., 100 000, are fully filled at the same time. We assume a realistic memory utilization in the range of hundreds of MB; however, a exact numbering is not possible due to many influencing factors, e.g., the deployed queue dropping policy.[40] The memory bandwidth must be at least as high as the total bandwidth, e.g., 200 Gbit/s.

Accounting

Due to legal constraints, all packets and bytes transmitted to the subscriber must be accounted before and after applying the HQoS queueing in order to register QoS-caused packet drops. After applying the HQoS functionality, packet loss is prevented by the HQoS mechanism of the BNG which is aware of all oversubscribed resources in the access network.

The total number of required counters is at least equal to the double of the number of queues but can be even higher. Therefore, assuming a number of 35 000 subscribers, 560 000 counters for packets, and 560 000 counters for bytes would be needed. Counting can be either performed within a P4 data plane before and after the HQoS system or by counters within the HQoS system.

3.3.2 | Upstream QoS

Rate limiting and prioritization for upstream traffic of each subscriber must performed by the residential gateway as this is the last networking device before the residential access line. However, a per subscriber upstream metering is required at the BNG in order to prevent abuse. In addition, all aggregated upstream flows of all subscribers must be queued at the BNG in a QoS aware way as well. In contrast to downstream traffic, all flows of a single QoS class from all subscribers can be combined in a single queue. By that, the required total queue memory size is much smaller than in downstream direction. Furthermore, most traffic is currently in downstream direction, and the BNG upstream link is typically no bottleneck. By that, only tiny buffers compared to downstream queues are needed and on-chip memory of most data center switches (typically 10–50 MB) will be sufficient.

3.4 | Fundamental functional tasks of a virtual BNG network function

Based on these requirements, the functional components of a BNG system can be derived which are listed in Table 2. Each functionality and their description correspond directly with one of the access service creation phases from Figure 2. The signs can be mapped later with the design in Figure 5.

TABLE 2 Functional components of BNG systems

Function	Sign	Description
Customer tunneling		Encapsulate customer traffic with PPPoE
Authentication, authorization, and accounting		IP address assignment, route establishment, and traffic counting
Traffic rate enforcement		Enforcing rate metering of upstream traffic and QoS class-dependent policies
Traffic access control		Provide processing of packets belonging to session authorized subscribers.
Traffic separation		Split control and data plane traffic, e.g., TTL = 0 packets
Quality of service		Provide techniques that allow prioritization of traffic
HQoS: queueing and hierarchical scheduling		Providing a set of queues (dependent on the QoS model) for each customer, hierarchical, and QoS aware scheduling
Service aggregation		Allow aggregation of the circuits from one or more access link platforms, virtual wire start point.
Security assurance		Provide protection of the system against misbehavior-like address spoofing, e.g., RPF

4 | CORD-BASED BNG ARCHITECTURE FOR PROGRAMMABLE DATA PLANES

In this section, we focus on the high-speed packet-header processing part, the data plane; the control plane part is beyond the scope of this article; however, we discuss the interfaces to that later. The overall system performance can only be effected by the control plane in case of session setups which is not the normal operation case. The performance evaluation of the following design would not be effected by that as for the evaluated scenarios, no control plane interaction is needed.

Our BNG design supports implementations for multiple P4 targets: (1) Barefoot Tofino, (2) Netronome SmartNIC, (3) P4-NetFPGA, and the (4) P4 behavioral model (bmv2). In the following, we propose an integration of programmable data planes into the CORD framework, fulfilling the functional requirements of BNGs by disaggregation the BNG functionality:

4.1 | Software architecture overview

One advantage over today's BNGs is the split of the functionality over three functional blocks, as depicted in Figure 4: a programmable data plane, the data plane controller, providing a control plane interface, and the BNG control plane itself. Furthermore, an access technology specific AN, e.g., an OLT for GPON, is required.

The functional requirements for the data plane are implemented in the *BNG header processing* component of the data plane, depicted in light green. The heterogeneity of different P4 data plane hardware implementations and their control plane interfaces is hidden by the hardware-specific data plane controller which provides a uniform interface towards the control plane.

The BNG data plane controller is running on the CPU of the programmable data plane device. The BNG control plane is operating as an x86-based VNF. Besides the programmable data plane, the control plane also controls the attached ANs. External components, such as the Radius server or the Operations Support System (OSS), are attached to the northbound interface of the BNG control plane.

4.2 | BNG pipeline design

On the basis of the CORD data center architecture and on current national requirements, which are forced by law, we propose a protocol stack and design architecture as depicted in Figure 5 (see Table 2 for the meaning of the symbols). The end-to-end header processing works as follows: the residential gateway (RG) sets up one PPPoE connection with the BNG. Different services are identified by multiple VLAN tags. The access line of a customer is expanded to the BNG by a VLAN-based virtual wire.

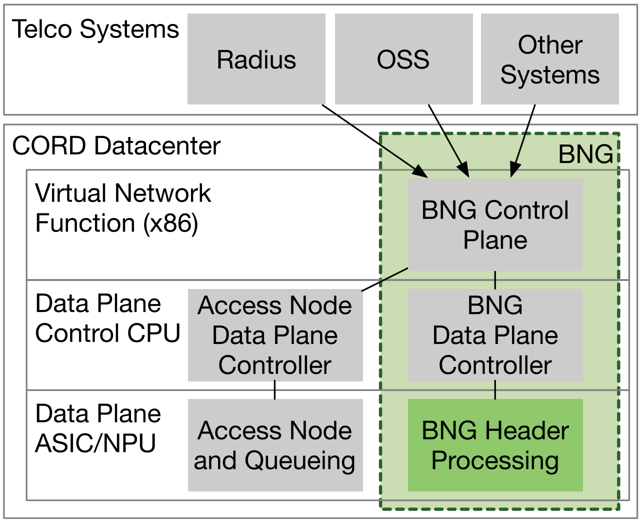
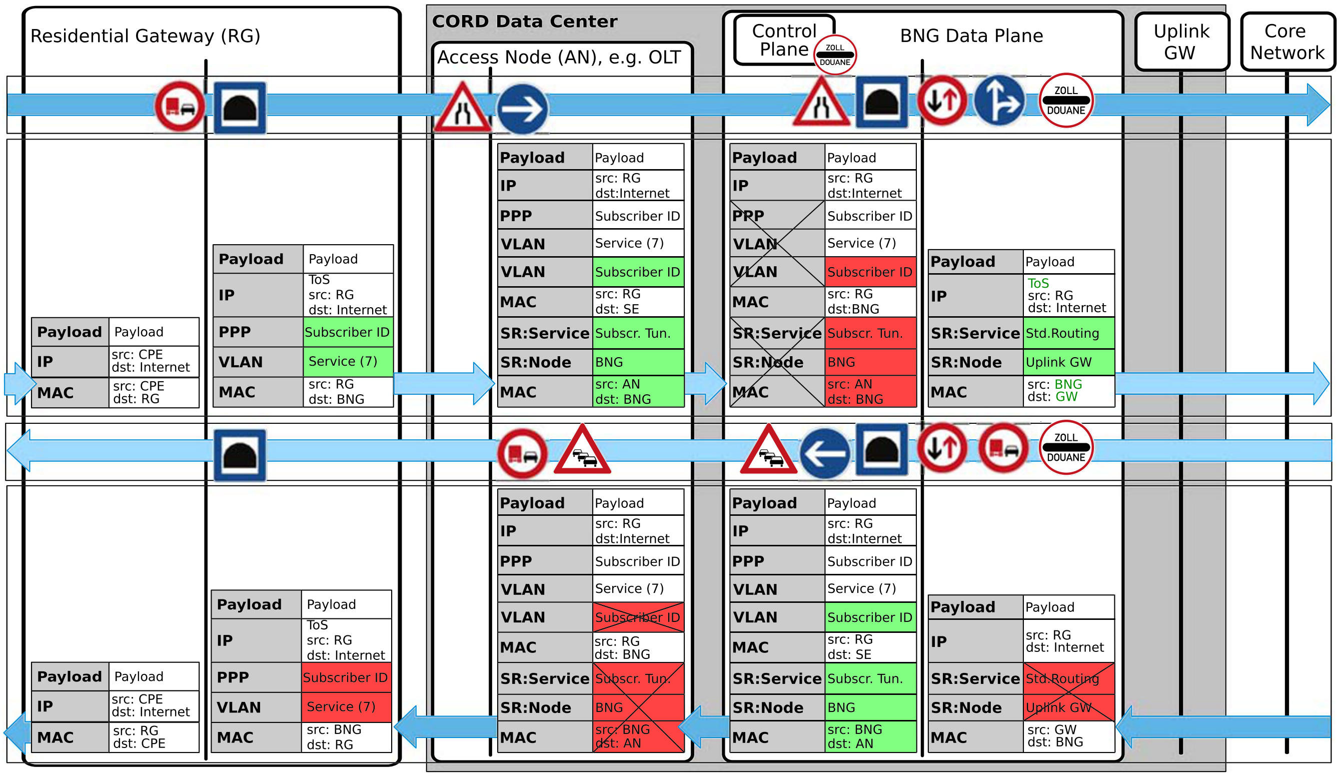


FIGURE 4 Disaggregated BNG, following the CORD-idea, consisting of a ASIC/NPU-based data plane for header processing and fully virtualized control plane stack running on commodity CPUs, typically x86 systems



Payload	Payload
IP	src: CPE dst: Internet
MAC	src: CPE dst: RG

Payload	Payload
IP	ToS src: RG dst: Internet
PPP	Subscriber ID
VLAN	Service (7)
MAC	src: RG dst: BNG

Payload	Payload
IP	src: RG dst:Internet
PPP	Subscriber ID
VLAN	Service (7)
VLAN	Subscriber ID
MAC	src: RG dst: SE
SR:Service	Subscr. Tun.
SR:Node	BNG
MAC	src: AN dst: BNG

Payload	Payload
IP	src: RG dst:Internet
PPP	Subscriber ID
VLAN	Service (7)
VLAN	Subscriber ID
MAC	src: RG dst:BNG
SR:Service	Subscr. Tun.
SR:Node	BNG
MAC	src: AN dst: BNG

Payload	Payload
IP	ToS src: RG dst: Internet
SR:Service	Std.Routing
SR:Node	Uplink GW
MAC	src: BNG dst: GW

Payload	Payload
IP	src: CPE dst: Internet
MAC	src: RG dst: CPE

Payload	Payload
IP	ToS src: RG dst: Internet
PPP	Subscriber ID
VLAN	Service (7)
MAC	src: BNG dst: RG

Payload	Payload
IP	src: RG dst:Internet
PPP	Subscriber ID
VLAN	Service (7)
VLAN	Subscriber ID
MAC	src: RG dst: BNG
SR:Service	Subscr. Tun.
SR:Node	BNG
MAC	src: BNG dst: AN

Payload	Payload
IP	src: RG dst:Internet
PPP	Subscriber ID
VLAN	Service (7)
VLAN	Subscriber ID
MAC	src: RG dst: SE
SR:Service	Subscr. Tun.
SR:Node	BNG
MAC	src: BNG dst: AN

Payload	Payload
IP	src: RG dst: Internet
SR:Service	Std Routing
SR:Node	Uplink GW
MAC	src: GW dst: BNG

FIGURE 5 Design for implementing BNG functionality in a CORD-like data center. The upper part describes the upstream direction, the lower part the downstream packets. The vertical bar separates the representations of ingoing and outgoing packets. Green header fields will be added; red header fields will be removed by the corresponding device

In addition, the packets are encapsulated in two MPLS segment routing headers and a new outer Ethernet header, as intended by the CORD fabric which forwards the packets from the AN to the BNG.

At session setup, the PPPoE control messages arriving at the BNG are forwarded to the control plane component. After authenticating the subscriber, a PPPoE session is created, and all required flow rules, enabling packet forwarding from and to the Internet, are installed in the BNG data plane. For normal traffic, the BNG removes the access header stack as well as the CORD MPLS headers, after applying reverse-path filtering and ACLs, and forwards the packet to the uplink interface with two new MPLS headers. Downstream traffic is processed accordingly in the opposite order.

In today's design, uplink rate limiting and traffic shaping are implemented in the residential gateway. Therefore, policing at the AN or BNG is sufficient for avoiding bandwidth violation.

In contrast to this, downstream traffic requires policing, traffic shaping, and prioritization at the central office in or close to the BNG. The queueing requirements of ISP access networks, e.g., huge buffers and hierarchical queues in large quantities, are not met by today's commodity packet switching ASICs. Therefore, this functionality can be either implemented on the AN or a subsequent FPGA as proposed in Section 5. ANs typically include a nonprogrammable switching chip which includes sufficient queueing capabilities for their own subscribers. As multiple of them are used in a CORD data center, we assume the queueing functionality to be placed there, as the resources are available anyway. However, due to a heterogeneous access network and different access technologies, queueing on ANs might not be possible in all scenarios. Thus, we will focus in the following on an FPGA-based queueing system.

5 | FPGA-BASED HIERARCHICAL QUEUEING AND QOS SUPPORT

While P4 is a very powerful language to describe packet header processing, it is not designed for describing packet queueing and scheduling behavior which is needed at ISP service edges in downstream direction. For that, as shown in Figure 6, an additional FPGA can be either subsequently or edgewise added to a P4-BNG implementation. By that, hardware, e.g., a P4 programmable switch, which is highly specialized for packet header processing and

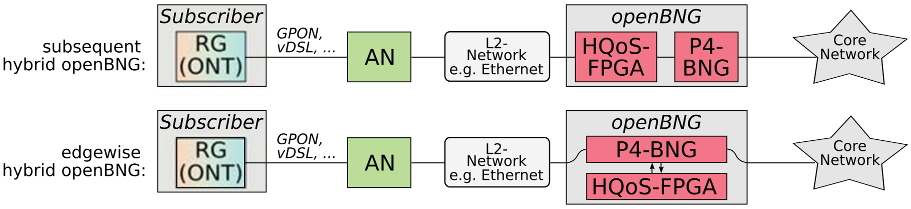


FIGURE 6 Realizing a openBNG with a programmable P4 data plane and a subsequent FPGA as traffic queueing and shaping processor. Both functional components, the P4 and FPGA chip, can be realized either within the same device or in subsequent appliances. The access network allows a heterogeneous topology with multiple technologies as vDSL or GPON

table lookups, can be used for most functionality. In addition, FPGAs, which allow realizing any boolean logic on the chip, can be used to implement the needed queueing behavior. For that, we suggest a design as depicted in Figure 7. While building upon available IP cores for Ethernet and DDR memory access, the queueing logic could be expressed without using 3third party IP cores in the hardware description language Verilog. However, it should be noted that such an implementation requires much more effort compared to high-level languages such as P4 which are specially build for a specific purpose. We believe that similar approaches than P4 could become beneficial and applicable for queueing and scheduling in future as low-level languages as Verilog and VHDL are very time-consuming, error-prone, and by that, a noteworthy hurdle.

The implemented design, which we will describe in the following briefly, works as follows: first, incoming packets will be classified by the FPGA and enqueued in one of the FIFO queues. For that, a packet classifier can be either implement within the FPGA only or within a preceding P4 switch. In the latter case, the queue ID, e.g, a 32-bit value, will be committed to the FPGA as an additional header in front of the existing outer Ethernet header. By that, the FPGA only extracts this queue ID and removes the leading ID bits of the packet. The advantage of this approach is that a queue and QoS class classification can be done with ease inside a P4 pipeline which is made exactly for such lookups.

Approaches for hierarchical and oversubscribed access networks exist in related work as well. Laki et al. proposed a per packet value approach which allows a QoS aware drop policy in oversubscribed access network.[41] Due to the accounting accuracy reasons and inter customer separation, this approach cannot be applied one by one at the moment. However, we consider this per packet value approach as very promising since it would simplify the HQoS scheduling in some points and could be realized within the openBNG design as well.

Last, we want to highlight the capabilities of FPGAs and programmable pipelines in the context of AQM. As most currently used Layer 4 transport protocols, mainly TCP, and associated congestion control mechanism tend to

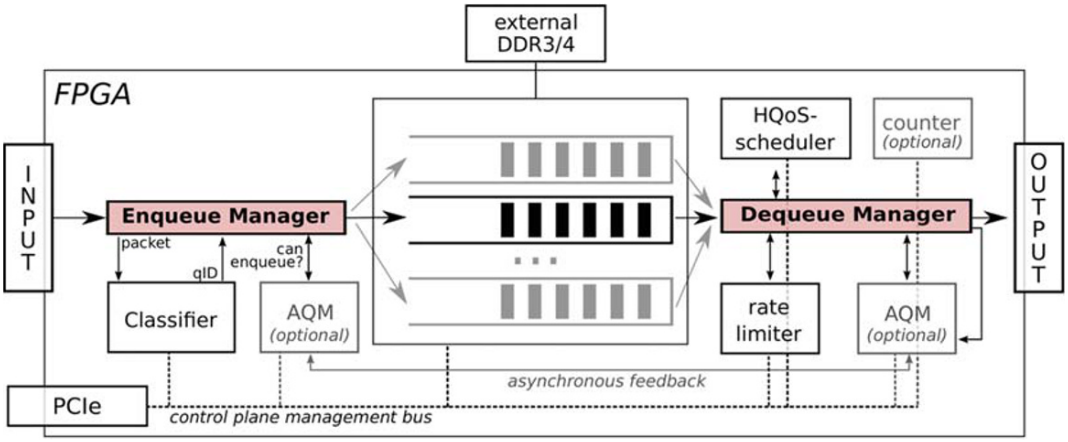


FIGURE 7 Design overview of an FPGA-based QoS and shaping processor. After being processed by the P4-BNG pipeline, packets will enter the FPGA on the *INPUT* side. The enqueue manager decides, supported by a classifier and an optional Active Queue Management (AQM) module, if and in which queue a packet should be inserted. While being queued, packets are stored in external DRAM. The dequeue manager decides, which packet should be sent next and if packets should be dropped from a queue by an AQM algorithm. Parameter configuration can be done via PCIe

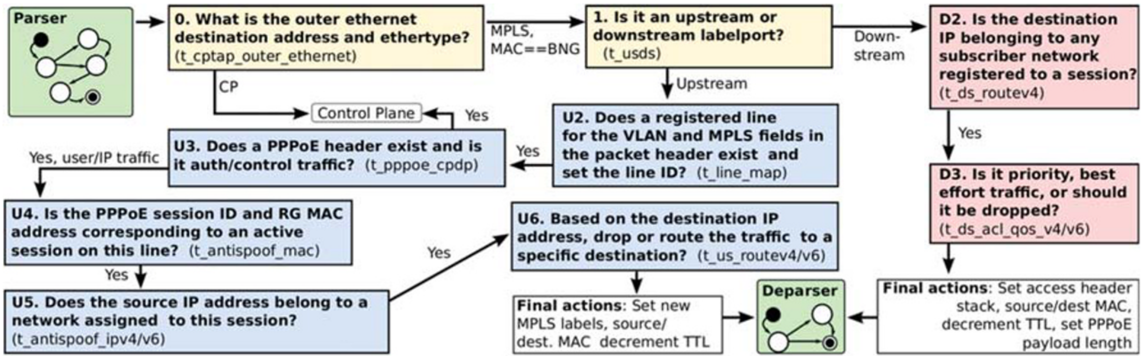

Parser
0. What is the outer ethernet destination address and ethertype? (t_cptap_outer_ethernet)
MPLS, MAC==BNG
1. Is it an upstream or downstream labelport? (t_usds)
Down-stream
D2. Is the destination IP belonging to any subscriber network registered to a session? (t_ds_routev4)
CP
Control Plane
Yes
Upstream
U2. Does a registered line for the VLAN and MPLS fields in the packet header exist and set the line ID? (t_line_map)
Yes
Yes, user/IP traffic
U3. Does a PPPoE header exist and is it auth/control traffic? (t_pppoe_cpdp)
Yes
D3. Is it priority, best effort traffic, or should it be dropped? (t_ds_acl_qos_v4/v6)
U4. Is the PPPoE session ID and RG MAC address corresponding to an active session on this line? (t_antispoof_mac)
Yes
U6. Based on the destination IP address, drop or route the traffic to a specific destination? (t_us_routev4/v6)
Yes
U5. Does the source IP address belong to a network assigned to this session? (t_antispoof_ipv4/v6)
Final actions: Set new MPLS labels, source/dest. MAC decrement TTL
Deparser
Final actions: Set access header stack, source/dest MAC, decrement TTL, set PPPoE payload length

FIGURE 8 Control flow of the P4-BNG data plane including packet parsing and deparsing. Yellow and green parts are applied on all packets, red boxes are downstream packet handling, and blue boxes are used for upstream traffic handling only

fill buffers until packet loss occur, huge buffer sizes will lead to a higher latency, called bufferbloat. Given that the optimal buffer size depends on many factors, e.g., RTT and number of flows, constant buffer sizing without bufferbloat is not possible. AQM algorithms as CoDel (RFC 8289) and PIE (RFC 8033) are state-full packet drop and marking algorithms which claim to manage the queue level utilization close to the optimal value.[42] While fixed data plane chips currently do not support these advanced drop policies, a programmable P4 pipeline[43] or FPGA can be used for them as well. First experiments with CoDel and PIE integrated in this design have shown very promising results but are not the focus of this work, and the feasibility on FPGAs has already been discussed by related work before.[31]

6 | P4-BASED PIPELINE IMPLEMENTATION

In this section, we describe our implemented P4 pipeline of a BNG data plane based on the requirements obtained in Sections 2 and 4, which is OpenSource available as part of the openCORD project on GitHub*. The implementation consists of (1) a parser implementation, explained in detail in Section 6.1, which enables splitting up the headers of incoming packets, and (2) a P4 programmable match-action pipeline.

Figure 8 provides an overview over the BNG pipeline modeled in P4. Note that the pipeline of a BNG differs significantly between upstream and downstream traffic processing. Thus, the depicted flow diagram can be separated into an upstream (light blue) and downstream (light red) part.

After parsing all incoming packets, the P4 programmable pipeline will be applied to the incoming packet. By that, the control traffic is forwarded to the control plane and identified in the table `t_cptap_outer_ethernet`. Then, a subsequent table `t_usds` decides based on the combination of physical port and *MPLS0* label, whether a packet comes from a core or an access port and is destined to the upstream or the downstream pipeline, described in Sections 6.2 and 6.3.

6.1 | Ingress parsing

In a first step after entering the P4-BNG, packets are parsed by the ingress parser. The incoming stream of bits are transformed by the parsing process in a meaningful internal data representation. This data representation, called packet header vector, can be accessed within the subsequent P4 match-action pipeline. Figure 9 depicts a sample P4 parser implementation which is able to understand all possible header stacks of incoming packets, already known from the design in Figure 5.

First, all possible packet headers of the BNG system must be specified in the P4 program which is depicted in Lines 1–29 which is similar to a `typedef` in C. Second, instances on the defined header types are created in the `struct`

*https://github.com/opencord/p4se

```
1   header ethernet_t {
2     bit<48> dstAddr;
3     bit<48> srcAddr;
4     bit<16> etherType; }
5
6   header vlan_t {
7     bit<16> vlanID;
8     bit<16> etherType; }
9
10  header mpls_t {
11    bit<20> label;
12    bit<3>  tc;
13    bit<1>  s;
14    bit<8>  ttl; }
15
16  header pppoe_t {
17    bit<4>  version;
18    bit<4>  typeID;
19    bit<8>  code;
20    bit<16> sessionID;
21    bit<16> totalLength;
22    bit<16> protocol; }
23
24  header bng_cp_t {
25    bit<16> stamp;
26    bit<32> fwd_port;
27    bit<48> eth_dstAddr;
28    bit<48> eth_srcAddr;
29    bit<16> eth_etherType;}
30
31  struct headers {
32    bng_cp_t    bng_cp;
33    ethernet_t  ethernet_outer;
34    mpls_t      mpls0;
35    mpls_t      mpls1;
36    ethernet_t  ethernet_inner;
37    vlan_t      vlan_subsc;
38    vlan_t      vlan_service;
39    pppoe_t     pppoe;
40    ipv4_t      ipv4;
41    ipv6_t      ipv6;}
42
43  struct ingress_md_t {
44    bit<2>  usds;
45    bit<32> line_id;
46    bit<8>  subsc_id; ... }
47  struct metadata {ingress_md_t ingress_md;}
48  parser_value_set mpls_0_accesslabels;
49
50  parser BngParser(packet_in packet,
51    out headers hdr,
52    out metadata meta,
53    out digest_data_t digest_data,
54    inout switch_meta_t switchmetadata) {
55
56    state start {
57      packet.extract(hdr.ethernet_outer);
58      transition select(hdr.ethernet_outer.etherType) {
59        ETHERTYPE_MPLS: parse_mpls;
60        ETHERTYPE_CP: parse_bng_cp;
61        default: accept; }}
62
63    state parse_mpls {
64      packet.extract(hdr.mpls0);
65      packet.extract(hdr.mpls1);
66      transition select(hdr.mpls0.label) {
67        mpls_0_accesslabels: parse_ethernet_inner;
68        default: parse_ip; }}
69
70    state parse_ethernet_inner {
71      packet.extract(hdr.ethernet_inner);
72      transition select(hdr.ethernet_inner.etherType) {
73        ETHERTYPE_VLAN: parse_vlan_subsc;
74        default: accept; }}
75
76    state parse_vlan_subsc {
77      packet.extract(hdr.vlan_subsc);
78      transition select(hdr.vlan_subsc.etherType) {
79        ETHERTYPE_VLAN: parse_vlan_service;
80        ETHERTYPE_PPPOE: parse_pppoe;
81        default: accept; }}
82
83    state parse_vlan_service {
84      packet.extract(hdr.vlan_service);
85      transition select(hdr.vlan_service.etherType) {
86        ETHERTYPE_PPPOE: parse_pppoe;
87        default: accept; }}
88
89    state parse_pppoe {
90      packet.extract(hdr.pppoe);
91      ... }
92  }
```

FIGURE 9 P4 code for parsing incoming packets at the BNG. All packet headers of interest are recursively extracted and stored for a later usage within the P4 pipeline. *Note*: The code contains minor simplifications of next header type determination

headers starting in Line 31. Note that one header type, e.g., *ethernet_t* or *vlan_t*, can be instantiated multiple times. Similar to that, metadata fields are specified as well.

Third, the parser control flow is described as a state machine. The starting point is always the state *start*, starting in Line 56. After extracting the outer Ethernet header in Line 57, based on the *etherType* of this packet, the next parser state is determined. In case of a subsequent MPLS header, the *etherType* would be *0x8847*, marked as *ETHERTYPE_MPLS* in the pseudo code. *ETHERTYPE_CP* would indicate packets from the control plane without an MPLS label stack. Thus, for typical data packets from and to subscribers, the next state would be *parse_mpls*, and two MPLS labels will be extracted. By this parser, all possible header stack combinations of the BNG scenario can be extracted. Note that this implementation is slightly simplified and is susceptible to malformed packets, e.g., only one existing MPLS label. The transition after extracting the two MPLS labels, specified in Line 67, is based on a `parser_value_set`. This value set can be filled at runtime, and by that, a set of MPLS can be defined which contains an inner Ethernet header within the MPLS payload. Last, the parser is terminated by the statement *transition accept*, which implies the packet to be further processed by the P4 match-action pipeline as depicted in Figure 8. All unparsed bits of a packet are defined as payload, independent on its actual content.

Similar to the ingress parsing, packets are deparsed before leaving the P4 device. While deparsing, all valid headers from the packet header vector will be prepended to the packet payload. This can be done implicitly in the ordering of the header struct definition in Line 31 or explicitly by an deparser implementation.

6.2 | Upstream pipeline

The upstream pipeline (Figure 8, light blue) first applies the table `t_line_map`. This table maps each combination of a physical port, *MPLS0*/*MPLS1* label, and a subscriber VLAN ID to a unique *line ID*.

If the line is legitimate, the table t_pppoe_cpdp determines whether a packet is a PPPoE control plane packet or a data plane packet for further processing; all unknown and illegitimate packets are dropped for security reasons. Typical packets destined to the controller are PPPoE discovery, PPPoE LCP protocol, or keep-alive packets. The decision, if a packet is control traffic, can be done based on the inner L2 destination address, the service VLAN Ethertype, and the *PPPoE protocol*type.

This table is preconfigured at startup, only requires a few entries (<16), and does not grow with the number of subscribers or networks.

After authenticating the subscriber via PPP, the residential gateway's MAC address, the negotiated PPPoE session ID, the service VLAN ID, and the line ID are used as input for the table t_antispoof_mac, which write the *subscriber ID* to the packet metadata. As a subscriber ID is unique per line, only the combination of line ID and subscriber ID can identify a subscriber.

After ensuring that only authorized subscribers can use the data plane, IP source address spoofing is prevented. To this end, we have to distinguish between IPv4 and IPv6 traffic. For every authenticated subscriber, networks must be added to the t_antispoof_ipv4 and t_antispoof_ipv6 tables which ensure that the source address matches the allowed address range. Packets without a valid source address are dropped.

The last ingress table, t_us_routev4/6, decides to which core interface and MPLS destination a packet should be sent towards its destination address. Then, the original header stack becomes replaced by the MPLS-SR/IP stack which is used in the core network. Finally, an egress table applies the correct outer destination and source MAC addresses, before the packet is sent out.

6.3 | Downstream pipeline

Compared to the upstream, the downstream direction is much simpler and requires fewer tables. If a packet enters the BNG, we determine the authenticated subscriber (line ID and subscriber ID) based on the destination IP address in table t_ds_routev4/6, which is filled with all networks of currently authenticated subscribers. If a match occurs, the corresponding action attributes contain all information, required to construct the MPLS/VLAN/PPPoE header stack and the output port. Furthermore, a per-subscriber meter is applied to ensure not exceeding downstream bandwidths. Finally, in the egress pipeline, the PPPoE payload length is calculated based on the IP header length field.

6.4 | State configuration model

In the following, we suggest a higher level control/data plane interface for a BNG providing access to the functionality of our implementation described before. We also sketch the P4 tables required to make the given modifications. The implementation of the control interface is provided in *p4_runtime_mgr.py*. Besides this generic implementation, a specific implementation or each target, which is called by this generic interface, is needed, e.g., for the P4-NetFPGA, SmartNICs, or the BMv2.

Just as one example, enableLabelPortAccess(port, mpls0_label, ourOuterMAC, peerOuterMAC) sets a combination of a physical port and outer MPLS label to be considered for receiving upstream traffic from and transmitting downstream traffic to the access network. Furthermore, the function sets the expected outer MAC address of the BNG and the communication partner's expected outer MAC address to the corresponding value, and it adds entries to the tables t_usds and table t_ds_srcmac. The insertion of the table entries is performed by the corresponding BNG data plane controller which has a unified interface to the control plane.

Table addUpstreamRouteV4/V6(..) is another example which sets a route for authenticated, reverse-path-filtered subscriber traffic to next hops in the core network. For a network prefix and a service ID, the next hop's MAC address and the MPLS segment routing labels can be defined. It is possible to specify NULL for the upstream route to deny traffic to a certain network, thus to implement an IP-based ACL. It adds an entry to table t_us_routev4/v6 for the downstream direction.

All other API methods, which are implemented and documented in the Open Source code as well, work very similar to those.

7 | VERIFICATION AND EVALUATION

The functionality of the BNG can be divided into two parts: (1) service creation and (2) packet forwarding from subscribers to the Internet described in the next two subsections. The first is executed only once per session, and therefore, its functional correctness is more important than its performance. The latter should be characterized by high throughout, low delay, and minimum packet loss. The sources of our P4-based BNG implementation and the verification framework data plane abstraction are available online.

7.1 | Functional verification

A BNG data plane implementation, targeted for production usage, must be verified for correct operation in various ways. In particular, the following desired qualitative behavior has been considered for testing:

- The control plane can send arbitrary packets on any port.
- On registered lines, subscriber authentication packets (PPPoE) are forwarded to the control plane.
- Subscriber traffic is forwarded by the data plane to a core-facing port if a PPPoE session is installed.
- *Reverse path forwarding* mechanisms are in effect. Packets with source addresses, subnets (Antispoof), or destination subnets (ACL) are dropped.

To validate this behavior, we implemented a Python-based framework based on the `Scapy` library for packet creation and sending. The verification framework runs on a separate computing node (Figure 10), sending packets and verifying the behavior. Although the maximum achievable bandwidth is low compared to high-performance load generators, it is sufficient for functional testing. In addition, the Packet Testing Framework (PTF) of the P4 consortium turned out to be very helpful and increases productivity. The tests are written in a target-independent way by introducing two abstraction layers: First, we propose a *control plane abstraction layer*for a simple BNG network function, based on the interface suggested in Section 6.4, which translates control plane commands into the underlying P4 state configuration commands (e.g., add an entry to a table). Example for control plane commands are adding subscriber circuits, activating/deactivating subscriber sessions after authentication/deauthentication, and adding network addresses and subnets to the subscriber sessions at runtime. Second, the framework consists of a P4 *data plane abstraction layer*, which we had to introduce because the state configuration interfaces for the P4 pipeline might be vendor specific. Unified control plane interfaces, e.g., P4-Runtime, will simplify this in future. All performed functional tests have shown the expected behavior, based on the capabilities of the target (see Section 7.5), and thus, BNG network functionality can be implemented with P4 targets. Performance characteristics are discussed in the following.

7.2 | P4-BNG performance characteristics

Our generic P4 Implementation can be executed by the software model *bmv2*, P4-NetFPGAs, and NPU-based Netronome P4-SmartNICs. As the *bmv2* is a software model for functional prototyping, a performance comparison to real hardware is not useful. Thus, we compare the Agilio SmartNICs and the P4-NetFPGA design with x86-based server

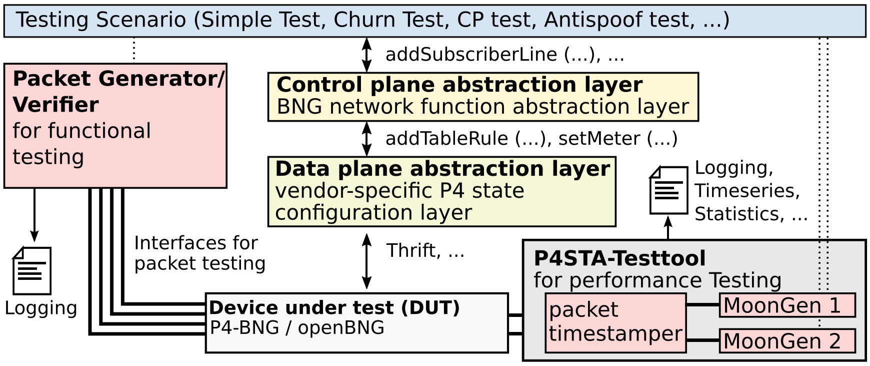

Testing Scenario (Simple Test, Churn Test, CP test, Antispoof test, ...)
addSubscriberLine (...), ...
Packet Generator/ Verifier
for functional testing
Control plane abstraction layer
BNG network function abstraction layer
addTableRule (...), setMeter (...)
Data plane abstraction layer
vendor-specific P4 state configuration layer
Logging, Timeseries, Statistics, ...
Logging
Interfaces for packet testing
Thrift, ...
P4STA-Testtool
for performance Testing
packet timestamper
MoonGen 1
MoonGen 2
Device under test (DUT)
P4-BNG / openBNG

FIGURE 10 Data plane verification framework with the abstraction layer used for the evaluation. The "Packet Generator/ Verifier" can be used for functional verification and control plane testing. The OpenSource P4STA-Testtool[44] is used for performance tests providing timestamp series for all test packets and packet loss information with a very high accuracy

which performs simple packet forwarding (without PPPoE termination or any other BNG functionality) in the Linux kernel by Docker containers. We assume that a Docker-based CORD[1] implementation (without kernel bypassing) of BNG functionality will perform, in the best case, similar to the Linux kernel in terms of throughput, packet loss, and latency. In addition, results for the P4 programmable Barefoot Tofino implementation, which is open source available as well, are given. Please note that the Tofino implementation supports additional functionality and is highly optimized on the chip hardware architecture.

The following metrics will be considered in the following:

- Reliability/packet loss of authenticated sessions
- The number of simultaneous supported sessions
- The total data plane performance (*throughput*, *latency*)
- The traffic is correctly limited based on meter settings

To investigate the performance, we used a testbed setup as depicted in Figure 10 based on the open source framework P4STA for load generation, aggregation, ant packet timestamping.[44] The device under test (DUT) is connected with two ports (core and subscriber side) to the P4STA packet timestamper, stamping all packets before and after the DUT with a timestamp inside the packet in a 16-byte field of the packet payload. Load generation is performed by two x86 servers with DPDK capable NICs and the Moongen load generator,[45] which create packets from the ANs (*MoonGen 1*) and the core side (*MoonGen 2*), as depicted in Figure 5. The P4STA framework provides the timestamp series for all packets after the test execution together with statistics on packet loss and preliminary evaluations. Due to link speed limitations of the NetFPGA-SUME and some SmartNICs, all tests are performed with 10 Gbit/s of link speed, even though some of the investigated SmartNICs would support up to 40 Gbit/s and the P4 programmable Tofino and the edgewise attached FPGA for QoS up to 100 Gbit/s.

As additional 30 bytes (inner Ethernet, 2xVLAN, PPPoE) are added to the packet in downstream direction, the downstream traffic must be shaped before the DUT in the programmable switch to 9.4 Gbit/s in order to prevent packet loss. Upstream traffic is not affected by that. The packet size before entering the DUT is 532 bytes (including timestamp header). We assume this packet size to be realistic in future as a mixture of TCP power flows (typically utilizing the MTU of 1500 byte fully) and many IoT, VoIP, and control traffic flows.

Figure 11 shows the average latency of the different target platforms depending on the throughput on a logarithmic scale for 512 subscribers. The latency of the P4-NetFPGA is very constant around 5 μ s, whereas the P4-SmartNIC, which processes the packets in many cores of the NPU, has a latency of 10 μ s for low bandwidths and increases up to 22 μ s for 10 Gbit/s. The latency of the Barefoot Tofino-based P4 switch, which consists of a high-performance packet pipeline with constant execution time, indicates no changes due to bandwidth variations. This latency is very constant slightly above $1\mu s$, including sending and receiving transceivers. As the link speed is manually reduced to 10 Gbit/s for comparison fairness reasons, already storing and forwarding a packet would take around 0.4 μ s. Thus, higher link speeds or smaller packet sizes would improve the measured latency strongly. In case of Linux kernel-based packet forwarding the latency for 10 Mbit/s is 53 μ s on average and goes down to 33 μ s for 100 Mbit/s. After reaching a bandwidth of 1700 Mbit/s, the Linux kernel is not able to handle more packets which leads to a rising packet loss rate and an average latency of 2.2 ms for 3000 Mbit/s and above. Increasing the number of subscribers to 4000 has no measurable impact.

The standard deviation of the latency is given in Table 3. The delay of a hardware pipeline, as the P4-NetFPGA or Barefoot Tofino, is very constant. In contrast to that, software approaches such as the Linux packet forwarding has a

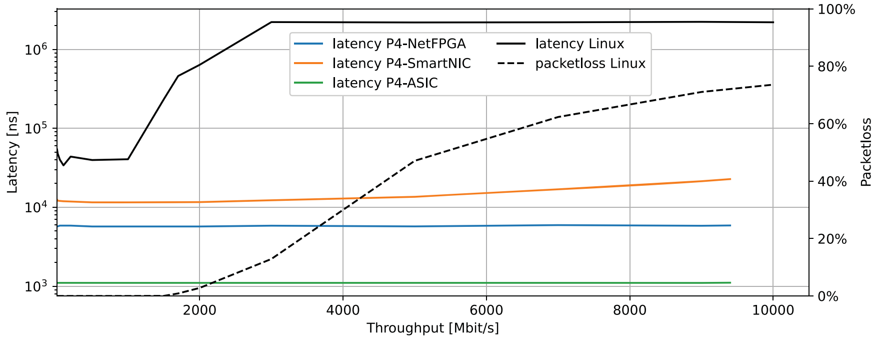


FIGURE 11 Latency for Linux kernel, P4-NetFPGA, and P4-SmartNICs depending on input rate for downstream traffic (512 subscribers). Packet loss for Linux kernel packet forwarding as dotted line. Packet loss of all P4 targets is constant 0. Packet size 532 bytes. A ingress rate of 9.4 Gbit/s is equivalent to an downstream egress rate of 10 Gbit/s as header stack increases

TABLE 3 Latency standard deviation of the P4-BNG implementation and Linux packet forwarding as reference

	Linux kernel	Netronome-SmartNIC	P4-NetFPGA	P4-ASIC
100 Mbit/s	13.2 μ s	1.7 μ s	<0.5 μ s	<10 ns
1000 Mbit/s	19.8 μ s	0.7 μ s	<0.5 μ s	<10 ns
9000 Mbit/s	157.9 μ s	1.6 μ s	<0.5 μ s	<10 ns

Note: Packet size is constant 532 bytes, and ingoing packet rate is shaped to 0.1, 1 and 9 Gbit/s.

higher variability in processing times because of many influencing factors but mainly the interrupt-based system architecture. NPUs, such as the evaluated P4-SmartNICs, are in between.

Table 4 lists the total latency (transceiver to transceiver) dependent on the number of subscribers for a constant downstream traffic rate of 9.4 Gbit/s, which is very constant and does not depend on the number of subscribers. The results for upstream traffic on SmartNICs and FPGA are slightly higher (hundreds of ns). We assume that this is caused by the more complex program (see Figure 8). As long as all flow tables are realized in SRAM and not in external DRAM, we expect similar behavior for all future programmable network devices as well. Rate limiting can be performed by Netronome SmartNICs and Barefoot Tofino, which have not shown any violations of the configured limits.

7.3 | End-to-end performance of a hybird FPGA/P4-ASIC OpenBNG

In the following, we present evaluation results for an edgewise hybrid BNG consisting of an FPGA and P4-ASIC, a Barefoot Tofino, as depicted in Figure 6. Packets entering the BNG in downstream direction will be first (1) processed by the P4-BNG, second (2) queued and scheduled by the FPGA, and third (3) sent by the FPGA through the P4 chip towards the subscriber. By that, the packet traverses the P4-ASIC twice which causes additional latency. Thus, we assume for a design with a subsequent FPGA, which is depicted in Figure 6 as well, even lower end-to-end latencies. However, the integration of a edgewise FPGA in the data path is easier as commodity FPGAs with a PCIe hardware accelerator form factor can be used. In addition, the total bandwidth of current (P4) programmable switches (>10 Tbit/s) is much higher than the required bandwidth in the BNG scenario and by that the total BNG bandwidth is not restricted by that. Furthermore, compared to typical subscriber RTTs, which are in the range of 10–100 ms, one additional μs for through passing the P4 switch twice is negligible. Packets in upstream direction can be processed by the P4-ASIC only and not additional queueing by the FPGA is needed.

Table 5 presents the measured results for a hybrid openBNG system with (1) no background load, (2) 1 Gbit/s of background load, and (3) 9 Gbit/s of background load. The latency measurements are performed with very little test load in order not to fill the queue of the FPGA. For that, in each test, 10 000 packets of 178-byte size, representing a low-latency VoIP packet which has currently the highest QoS requirements regarding latency, are sent through the BNG system and captured before and after through passing by the P4STA framework. First, we would like to highlight that no packet loss was observed in all tests as long as the configured queue bandwidth was never exceeded. Second, bloating one or multiple other queues with packets has negligible impact on the latency, caused by the hierarchical scheduler implementation.

In addition, we would like to highlight the traffic shaper accuracy of an FPGA. Even though 100 000 queues are realized on the FPGA, the rate jitter of a single subscriber is very little. For this test, the subscriber queue is configured to 50 000 Mbit/s. A UDP load generator, configured to 100 000 Mbit/s, will permanently fill the queue as it never

TABLE 4 P4-BNG latency without traffic shaping depending on number of subscribers for 9.4 Gbit/s of downstream traffic, packet size: 532 byte

#Subscribers	32	64	128	256	512	1024	2048	4000	std.dev.(n=4000)
P4-NetFPGA	5.08 μs	5.08 μs	5.08 μs	5.08 μs	5.08 μs	5.08 μs	5.08 μs	5.08 μs	242 ns
P4-SmartNIC	22.12 μs	22.13 μs	22.13 μs	22.14 μs	22.15 μs	22.19 μs	22.03 μs	22.06 μs	2.6 μs
P4-ASIC	1.17 μs	1.17 μs	1.17 μs	1.17 μs	1.17 μs	1.17 μs	1.17 μs	1.17 μs	1.54 ns

Note: All targets are limited, due to comparison fairness reasons, to 10 Gbit/s of link speed. The measurement results for the P4-ASIC are in the range of the measurement error and depend on the exact measurement surroundings as fiber cable length, packet pacing, and packet size.

TABLE 5 End-to-end latency of a FPGA/P4-ASIC hybrid openBNG system for 178-byte probe packets (VoIP packet size)

background load	# packets	loss	min latency	max. latency	average latency	latency std. dev.
none	10,000	0	2.02μs	12.85μs	7.24μs	3.00μs
1 Gbit/s	10,000	0	2.01μs	12.76μs	7.39μs	3.00μs
9 Gbit/s	10,000	0	2.01μs	14.47μs	7.34μs	3.19μs

Note: Background load is varied between 0 and 9 Gbit/s in total for all other subscribers with variable packet size. Measurement is performed in each test for 10 000 packets for a single subscriber with a rate of 2000 packets/s, not utilizing its configured bandwidth. Measurement queue and background load share hierarchical resources and by that shared rate limiters.

becomes empty. Figure 12 depicts the distribution of the interpacket arrival times. As you can see, the packets are close to uniformly distributed. For comparison, a commodity TCP stack on Linux systems would cause microbursts in the range of hundreds to thousands packets on a 10G/40G link[46] as fine-grained rate limiting in software would require a CPU interaction. However, the usage of rate-limiting capabilities in commodity network interface cards would lead to much better results but does not allow large-scale per-flow queueing.

7.4 | Resource utilization

Although the different target platforms have different hardware architectures, we provide some values in order to point out the bottlenecks. The P4-NetFPGA project is based on a Xilinx Virtex-7 690T chip, introduced in 2010, which persists mainly on many reconfigurable lookup tables (LUTs) and memory blocks (BRAM). The resource utilization of the P4-NetFPGA with 4096 subscribers is given in Table 6 and shows the total consumption of the design and the P4 pipeline only. This difference is caused by four 10G Ethernet IP cores, a microprocessor and additional peripheral logic. The limiting resource is BRAM, which is only available in selected areas of the FPGA and thus a utilization of 100% becomes very hard to achieve due to timing constraints.

The resource utilization of an FPGA-based QoS and shaping functionality is shown in Table 7 for two FPGA chip generations. In contrast to the resource utilization of the P4-NetFPGA project for a P4-BNG pipeline, most logic resources (LUTs) are used for I/O functionality as Ethernet IP-Cores, DDR3/4 memory controllers and PCIe infrastructure. However, HQoS with many queues require stateful information for each of these queues and therefore a lot of on-chip memory (BRAM an URAM).

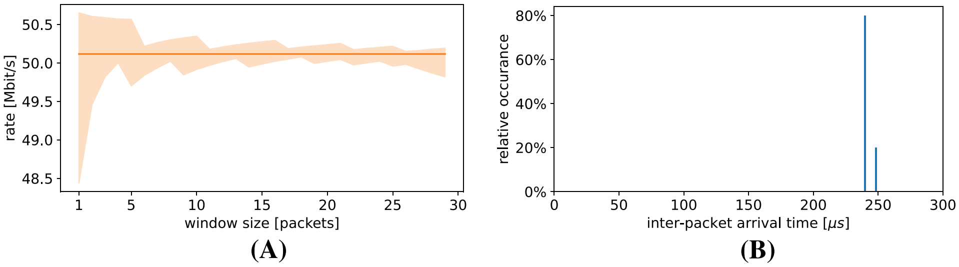


FIGURE 12 Interpacket arrival time (IPAT) distribution of an FPGA-based traffic shaper with a configured rate of 50 Mbit/s and 100 000 queues. The queue never becomes empty during the test, (a) depicts the observed rate over a window of n packets in average and 99.9% quantile, and (b) illustrates the distribution of the IPAT

TABLE 6 Resource utilization of a P4-NetFPGA (build upon the NetFPGA-SUME board) based BNG implementation without traffic shaping functionality for 4096 subscribers

	Total	P4 data path	Available
LUT	202 155 (47%)	163 013 (38%)	433 200
BRAM	1074.5 (73%)	952 (65%)	1470

TABLE 7 Resource utilization of a Xilinx UltraScale + FPGA (VCU1525 – XCVU9P) and the NetFPGA-SUME board (Xilinx Virtex 7 XC7VX690T) implementing HQoS support for 100 000 queues accepting up to 25 MB of packets and three hierarchical scheduling layers without the BNG data plane implementation and no packet counters

	NetFPGA-SUME (10G)			Xilinx VCU1525 (100G)		
	total	queueing logic	available	total	queueing logic	available
LUT	47,160 (10.89%)	6,463 (1.49%)	433,200	51,806 (4.38%)	5,494 (0.46%)	1,182,240
BRAM	355 (24.15%)	334 (22.7%)	1,470	92 (4.26%)	22 (1.02%)	2,160
URAM	*not available*		0	29 (3.02%)	29 (3.02%)	960
DSP	3 (< 1%)	3 (< 1%)	3,600	6 (< 1%)	3 (< 1%)	6,840
SerDes	2 (5.56%)	0 (0%)	36	5 (7%)	0 (0%)	76

Note: One external DDR3/DDR4 memory is used, and packet I/O is realized on the same Ethernet port for rx and tx. PCIe Gen3x1 is used for queue parameter configuration.

This is one reason for implementing the HQoS functionality on a state-of-the-art FPGA with 100 Gbit/s of link speed and more memory resources as well. However, for both FPGA types, a lot of on-chip memory blocks, concrete midsize BRAM, and larger URAM cells are needed and are by far the most consuming resource.

The total memory consumption is linear dependent on the number of queues, total capacity of the queueing system in packets, and the number of counters, e.g., dependent on the concrete implementation:

$$memory \propto 2xQueues + 2xMaxQueuedPackets + Counters. \tag{1}$$

Considering the resource utilization of the queueing logic only for the NetFPGA-SUME, this could be technically combined with a P4 pipeline on a single FPGA. However, due to the very high BRAM utilization, we assume that only a small scale BNG consisting of P4 pipeline and HQoS system would fit on a single FPGA. Indeed, multiple P4 to FPGA compiler vendors announced resource utilization improvements, and in addition, the NetFPGA-SUME platform does not reflect the resource availability of nowadays FPGAs. Newer FPGA generations from Intel and Xilinx provide much more resources, as also shown in the table, and by that, even more subscribers could be realized.

The resource utilization of our P4 service edge implementation for Barefoot Tofino, given in Table 8, shows that the number of subscribers is not the only influencing factor of the resource consumption. Depending on further requirements and parameters, the previously named goal of 35 000 subscribers per BNG[38] can be achieved with additional optimization of the P4 program.

All in all, we assume a combination of a P4-BNG and HQoS system on the same FPGA would require less resources than the sum of both systems and would be feasible. Therefore, a hybrid openBNG consisting of two FPGA chips would not be meaningful. However, the combination of a P4 programmable ASIC or NPU and an FPGA with QoS functionality is, to the best of our knowledge, an even more reasonable system concept.

7.5 | P4 language and target-specific limitations

Although P4 was designed as a target-independent language, we observed that modifications of the P4 code are needed. Netronome SmartNICs are able to execute P4_14 code, written for software reference switch BMv2, without modifications; all needed P4 functionalities were supported. First experiments with P4_16 have shown similar results. However, we observed limitations, which can lead to dropped flow rules regardless to the configured table size under special circumstances if the number of installed flow rules becomes too high. Barefoot Tofino has also shown a

TABLE 8 Resource utilization for Barefoot Tofino

	4096 subscribers	8192 subscribers
SRAM	12.81%	15.94%
TCAM	15.97%	15.97%
#Pipeline stages	9	9

very good support of the language and its features; however, the P4 code requires minor modifications. By further target-specific optimizations, the maximum number of subscribers could be improved further.

The P4-NetFPGA project, based on the Xilinx SDNet toolchain, supporting only P4_16 , does currently not support P4 tables with longest prefix match or multiple match inputs and *parser_value_sets*, as introduced in Section 6.1, which are not supported. As this is no limitation of FPGAs itself, this might be supported with future compiler releases. One benefit of P4 on FPGAs are external functions which can be used to integrate logic not supported by the P4 compiler, e.g., metering and traffic shaping. Ibanez et al. highlight an improvement of the P4 compiler toolchain used by the P4-NetFPGA project for future compiler versions,[26] which underlines our speculation of future compiler improvements. In addition, it should be mentioned that third party P4 compilers are available for FPGAs as well. However, we did not investigate them further, but we believe that this vendor disaggregation would be beneficial for technical progress towards powerful and easy programmable FPGAs.

Another interesting research field is the breakdown of one physical P4 programmable pipeline into multiple logical data plane applications. The BNG use case could benefit from this additional flexibility in terms of data plane sharing with other applications, e.g., mobile access networks, but also for in operation updates. Current approaches already have shown a feasibility of multitenant data planes at compile time[47] by composing multiple P4 programs. This approach allows, besides the realization of multiple data plane applications, the chaining of multiple network functions by packet recirculation. However, a partial reconfiguration of one data plane application at runtime without affecting other running applications would increase flexibility even further.

7.5.1 | Programming schedulers

The fact that P4 was made for packet header processing and not for scheduling led us to the usage of FPGAs for HQoS. Our approach based on the principle that QoS-aware flow classification can easily be done in P4. All other QoS functionalities were realized within the FPGA. Although in existing P4 data plane chips scheduler behavior configuration is currently target dependent and outside of the scope of P4, the language provides a mechanism (*intrinsic metadata*) allowing P4 programs to interface with any vendor-specific queueing and scheduling component. We observed a similar situation for multicast traffic, managed by vendor-specific logic which can be controlled by *intrinsic metadata* as well.

As discussed in Section 2.6, there are already first approaches for programmable schedulers proposed by academia. However, until now, these ideas are not yet useful in a common way, and therefore, no programmable QoS-ASICs exists.

On the basis of our findings while implementing a FPGA-based HQoS system, we share the opinion of related work that a composer of fixed functionality blocks would be a good way. Such blocks could be (1) FIFO queues, (2) rate limiter, (3) stateful registers, (4) ALUs, and other basic building blocks which can be each configured and composed. For that, a high-level language similar to P4 would be beneficial. In addition, we would like to mention the possibility of FPGA-like structures for scheduling within the queueing component consisting of a set of universal usable FIFO queues. On the one hand, the price of scheduling algorithms with very high complexity realized on flexible programmable hardware is a dross in bandwidth. On the other hand, scheduling is typically always within the scope of one single egress port, and by that, a processing rate of the link speed, e.g., 100 Gbit/s, would be sufficient. However, even that is a nontrivial problem.

Another bandwidth challenging issue is the packet buffer memory bandwidth. Our approach based upon external DDR3/4 memory which would not scale for ultrahigh bandwidth switches utilizing all ports. With internal high bandwidth DRAM memories in FPGAs or ASICs, this limitation could be shifted a little bit higher. In addition, external DDR memory would require many I/O ports which cannot be used for Ethernet links and by that lower the total backplane bandwidth of the switch. Furthermore, for many applications, deep packet buffers are not required but would require additional costs during production.

8 | CONCLUSION

The P4 pipeline description language is a powerful instrument that allows network designers creating highly functional and versatile data plane programs. In this article, we have proposed a P4-based implementation of a BNG network function data plane, which complies with all essential requirements of a large operator of a telecommunications network. In addition, we presented a hybrid data plane design consisting of a P4-ASIC and a FPGA for QoS and queueing which allows to create a fully fledged BNG with commodity hardware.

Together with our implementation, we have also proposed a vendor- and hardware-independent runtime configuration interface for a BNG data plane, allowing control planes to be decoupled from the latter. The latest advances in reconfigurable hardware enable the execution of the P4-BNG implementation and FPGA-based HQoS solution with high performance, demonstrated by our evaluation results. Following the open-source idea, we have shared the code with the networking community.

Next steps will be an in-depth integration and operational testing towards a flexible, programmable BNG network function with highest performance in productive use. Future enhancements of P4 or a similar language based on the same domain-specific high-level language concepts towards describing queueing and QoS as well would empower the language further and the BNG use case will benefit from that.

ACKNOWLEDGEMENTS

This article is devoted to our friend and coworker Wilfried Maas, who to our great dismay passed away while we finished this article. His excellent expertise on IP QoS enabled us to perform this work. We do miss his knowledge, friendliness, and humor every day. This work has been supported by Deutsche Telekom through the Dynamic Networks 6, 7, and 8 projects, and in parts by the German Research Foundation (DFG) as part of the project C2 within the Collaborative Research Center (CRC) 1053 - MAKI. Furthermore, we thank our colleagues for their valuable input and feedback. Open access funding enabled and organized by Projekt DEAL.

ORCID

Ralf Kundel https://orcid.org/0000-0003-1711-5990

REFERENCES

1. Peterson L, Al-Shabibi A, Anshutz T, et al. Central office re-architected as a data center. *IEEE Commun Mag*. 2016;54(10):96-101.
2. Nobach L, Blendin J, Kolbe H-J, Schyguda G, Hausheer D. Bare-metal switches and their customization and usability in a carrier-grade environment. In: Conference on Local Computer Networks (LCN) IEEE; 2017:649-657.
3. Bosshart P, Daly D, Gibb G, et al. P4: programming protocol-independent packet processors. *ACM SIGCOMM Comput Commun Rev*. 2014;44(3):87-95. http://doi.acm.org/10.1145/2656877.2656890
4. Kundel R, Nobach L, Blendin J, et al. P4-BNG: central office network functions on programmable packet pipelines. In: 15th International Conference on Network Service Management (CNSM) IEEE/IFIP; 2019:1-9.
5. Anschutz T. TR-101: migration to Ethernet-based broadband aggregation. *Technical report*. 101 Issue 2, Broadband Forum; 2011.
6. Cui A, Hertoghs Y. TR-145: multi-service broadband network functional modules and architecture. *Technical report*. 145 Issue 1, Broadband Forum; 2012.
7. Alter C, Hertoghs Y, Li H, Rius i Riu J. *TR-178: multi-service broadband network architecture and nodal requirements*. 2014;178 Issue 1.
8. Hu F, Hua R, Hu S. Yang data model for configuration interface of control-plane and user-plane separation BNG. IETF Internet-Draft; 2018.
9. Ooghe S, Voigt N, Platnic M, Haag T, Wadhwa S. Framework and requirements for an access node control mechanism in broadband multi-service networks. IETF, Internet Requests for Comments, RFC 5851; 2010.
10. Shrum E. TR-092: broadband remote access server (BRAS) requirements document. *Technical report*. 092 Issue 1, Broadband Forum; 2004.
11. Haleplidis E, Pentikousis K, Denazis S, Salim JH, Meyer D, Koufopavlou O. Software-defined networking (sdn): Layers and architecture terminology. IETF, Internet Requests for Comments, RFC 7426; 2015.
12. John W, Devlic A, Ding Z, et al. Split architecture for large scale wide area networks. arXiv preprint arXiv:14022228; 2014.
13. McKeown N, Anderson T, Balakrishnan H, et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Comput Commun Rev*. 2008;38(2):69-74.
14. Chiosi M, Clarke D, Willis P, et al. Network functions virtualisation: an introduction, benefits, enablers, challenges & call for actions. SDN and OpenFlow World Congress, https://portal.etsi.org/nfv/nfv_white_paper.pdf; 2012.
15. Merkel D. Docker: lightweight linux containers for consistent development and deployment. *Linux J*. 2014;2014(239):2. http://dl.acm.org/citation.cfm?id=2600239.2600241
16. Sefraoui O, Aissaoui M, Eleuldj M. Openstack: toward an open-source solution for cloud computing. *Int J Comput Appl*. 2012;55(3):38-42.
17. Berde P, Gerola M, Hart J, et al. ONOS: towards an open, distributed SDN OS. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN) ACM; 2014. http://doi.acm.org/10.1145/2620728.2620744
18. Timmers M, Guenach M, Nuzman C, Maes J. G.fast: evolving the copper access network. *IEEE Commun Mag*. 2013;51(8):74-79.
19. Emmerich P, Raumer D, Gallenmúller S, Wohlfart F, Carle G. Throughput and latency of virtual switching with open vswitch: a quantitative analysis. *J Netw Syst Manag Springer*. 2018;26(2):314-338.
20. Budiu M, Dodd C. The P416 programming language. *ACM SIGOPS Oper Syst Rev*. 2017;51(1):5-14.

21. P4-to-VHDL: automatic generation of high-speed input and output network blocks. *Microprocess Microsyst*. 2018;56:22-33.
22. Neshatpour K, Malik M, Ghodrat MA, Sasan A, Homayoun H. Energy-efficient acceleration of big data analytics applications using FPGAS. In: 2015 IEEE International Conference on Big Data (big data) IEEE; 2015:115-123.
23. Nobach L, Hausheer D. Open, elastic provisioning of hardware acceleration in nfv environments In International Conference on Networked Systems (NetSys); 2015:1-5.
24. Bronstein Z, Roch E, Xia J, Molkho A. Uniform handling and abstraction of NFV hardware accelerators. *IEEE Netw*. 2015;29(3):22-29.
25. Mijumbi R, Serrat J, Gorricho J-L, Bouten N, De Turck F, Boutaba R. Network function virtualization: state-of-the-art and research challenges. *IEEE Commun Surv Tut*. 2016;18(1):236-262.
26. Ibanez S, Brebner G, McKeown N, Zilberman N. The p4-> NetFPGA workflow for line-rate packet processing. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays; 2019:1-9.
27. Xiong Z, Zilberman N. Do switches dream of machine learning? Toward in-network classification. In: Workshop on Hot Topics in Networks(HotNets) ACM; 2019:25-33.
28. Ricart-Sanchez R, Malagon P, Salva-Garcia P, Perez EC, Wang Q, Calero JMA. Towards an FPGA-accelerated programmable data path for edge-to-core communications in 5G networks. *J Netw Comput Appl*. 2018;124:80-93.
29. Singh SK, Rothenberg CE, Patra G, Pongracz G. Offloading virtual evolved packet gateway user plane functions to a programmable ASIC. In: CoNEXT Workshop on Emerging in-Network Computing Paradigms (ECNP) ACM; 2019:9-14.
30. Shah R, Kumar V, Vutukuru M, Kulkarni P. TurboEPC: leveraging dataplane programmability to accelerate the mobile packet core. In: Proceedings of the Symposium on SDN Research, SOSR 2020. ACM ACM; 2020; New York, NY, USA:83-95. https://doi.org/10.1145/3373360.3380839
31. Sivaraman A, Winstein K, Subramanian S, Balakrishnan H. No silver bullet: extending SDN to the data plane. In: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII ACM; 2013; New York, NY, USA. https://doi.org/10.1145/2535771.2535796
32. Sivaraman A, Subramanian S, Alizadeh M, et al. Programmable packet scheduling at line rate. In: Proceedings of the 2016 ACM Sigcomm Conference, SIGCOMM 2016 ACM; 2016; New York, NY, USA:44-57. https://doi.org/10.1145/2934872.2934899
33. Alcoz AG, Dietmüller A, Vanbever L. SP-PIFO: approximating push-in first-out behaviors using strict-priority queues. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20) USENIX; 2020; Santa Clara, CA:59-76. https://www.usenix.org/conference/nsdi20/presentation/alcoz
34. Sharma NK, Zhao C, Liu M, et al. Programmable calendar queues for high-speed packet scheduling. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20) USENIX; 2020; Santa Clara, CA:685-699. https://www.usenix.org/conference/nsdi20/presentation/sharma
35. Agilent Technologies I. Understanding DSLAM and BRAS access devices. White Paper; 2006.
36. Bifulco R, Dietz T, Huici F, et al. Rethinking access networks with high performance virtual software brases. In: Second European Workshop on Software Defined Networks (EWSDN) IEEE; 2013:7-12.
37. Consortium TS. Split architecture for large scale wide area networks - deliverable D3.3. Accessed: 2020-01-30 https://arxiv.org/abs/1402.2228; 2012.
38. Rodriguez D. Next generation central offices transform network edge with datacenter economics, cloud flexibility. Blog post; 2018. Accessed: 2018-05-14.
39. Grazia CA, Patriciello N, Klapez M, Casoni M. A cross-comparison between tcp and aqm algorithms: Which is the best couple for congestion control? In: 14th International Conference on Telecommunications (ConTEL) IEEE; 2017:75-82.
40. Kundel R, Wallerich J, Maas W, Nobach L, Koldehofe B, Steinmetz R. *Queueing at the Telco Service Edge: Requirements, Challenges and Opportunities*. Stanford, US: Workshop on Buffer Sizing; 2019. http://buffer-workshop.stanford.edu
41. Laki S, Gombos G, Hudoba P, et al. Scalable per subscriber QoS with core-stateless scheduling. In: SIGCOMM 2018 Demo ACM; 2018: 84-86.
42. Jiang H, Wang Y, Lee K, Rhee I. Tackling bufferbloat in 3G/4G networks. In: Internet Measurement Conference (IMC) ACM; 2012: 329-342.
43. Kundel R, Blendin J, Viernickel T, Koldehofe B, Steinmetz R. P4-codel: active queue management in programmable data planes. In: Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) IEEE; 2018:1-4.
44. Kundel R, Siegmund F, Blendin J, Rizk A, Koldehofe B. P4STA: high performance packet timestamping with programmable packet processors. In: Network Operations and Management Symposium (NOMS) IEEE/IFIP; 2020:1-9.
45. Emmerich P, Gallenmüller S, Raumer D, Wohlfart F, Carle G. Moongen: a scriptable high-speed packet generator. In: Internet Measurement Conference(IMC) ACM; 2015:275-287.
46. Kundel R, Rizk A, Koldehofe B. Microbursts in software and hardware-based traffic load generation. In: Network Operations and Management Symposium (NOMS) IEEE/IFIP; 2020:1-2.
47. Wu D, Chen A, Ng TSE, Wang G, Wang H. Accelerated service chaining on a single switch ASIC. In: Proceedings of the 18th acm workshop on hot topics in networks, HotNets 2019 ACM; 2019; New York, NY, USA:141-149. https://doi.org/10.1145/3365609.3365849

AUTHOR BIOGRAPHIES

Ralf Kundel is a third year PhD student at Technical University of Darmstadt where he received his master's degree in 2017. He is working in joint projects with Deutsche Telekom towards applying open hardware in carrier grade access networks. His research interests are programmable hardware, especially FPGAs and P4-programmable ASICs, theory and management of packet buffers, data plane virtualization, and NFV offloading.

Leonhard Nobach is a Senior Expert at Deutsche Telekom Technik GmbH for access networks. He obtained his PhD degree at the Technical University of Darmstadt in 2018, where he received his master's degree in 2012 as well. During this PhD study, Mr. Nobach also worked as a PhD intern at Nokia Bell Labs for 4 months. Besides packet switching hardware, his areas of expertise include NFV state migration, virtualization performance, NFV hardware acceleration, software-defined networks, communication security, and mobile peer-to-peer networks.

Jeremias Blendin is a software engineer at Barefoot Networks, an Intel company. Before that, he worked as a researcher at DE-CIX Management GmbH on the world's largest Internet exchange point. He received his PhD degree from the Department of Electrical Engineering and Information Technology at Technical University of Darmstadt in 2018 and his Diplom in Wirtschaftsinformatik in 2013 at the same university. His work focuses on programmable network data planes.

Wilfried Maas obtained his Dipl. Ing. degree at university of applied sciences Dieburg in 1988. Since 1991, he had been working at Deutsche Telekom in various positions as Senior Network Engineer. His expertise includes deep knowledge in access networks, core networks, and interconnection of autonomous systems. Particularly noteworthy is deep knowledge and experience in carrier grade quality of service (QoS). To our great dismay, Wilfried passed away while we finished this work. We lost a great friend and coworker. This paper is devoted to him.

Andreas Zimber is a senior expert for IP access networks at Deutsche Telekom. He obtained his diploma in Electrical Engineering in 2010 from the university of applied sciences Darmstadt. Since 2010, he is working at Deutsche Telekom. During that time, he took part in the development and integration of multiple generations of broadband network gateways (BNGs). Andreas is a specialist for MPLS-routed networks, including multicast packet distribution, quality of service (QoS), and residential access networks.

Hans-Joerg Kolbe heads the engineering organization at Deutsche Telekom that builds SDN-based access networks. Further, as Chief Engineer, he is responsible for the system design of DT's Access 4.0 program as chief product owner. Before joining Deutsche Telekom, he headed various research groups and a presales team at NEC Laboratories Europe. Prior to that, he was responsible for the broadband access network architecture and service implementation at Arcor AG, the German fixed line division of Vodafone. Hans-Joerg has 20 years of experience in broadband network technology and works on SDN and virtualization since 2010. He has significantly contributed to multiple ETSI and BBF standards as well as scientific papers and (co-) authored 17 granted patents. He holds a PhD in physics from University of Marburg, Germany.

Georg Schyguda is Squad Lead Inventory & Communications at Deutsche Telekom Technik GmbH. He obtained his Diploma in Sociology and Information Science in 1991 at the Technical University of Darmstadt. During his career, he covered several positions in T-Systems, Deutsche Telekom R&D unit, and currently he works at Deutsche Telekom Technik. His expertise covers portfolio management for carrier grade services and platforms, inventory management, process optimization, cost engineering, and the management of R&D resources for Deutsche Telekom Technik.

Vladimir Gurevich is a Principal Engineer at Intel. He has been involved in the design and development of network drivers and other technologies since the early 90s, working at leading engineering roles at companies like Cisco Systems, Broadcom Corporation and Barefoot Networks. He is also known as a professional educator, responsible for creating Barefoot Academy and teaching P4 programming to the hundreds of engineers and researchers world wide. Vladimir received his MSc is Applied Mathematics at the Moscow Institute of Radio Engineering, Electronics and Automation (MIREA, Technical University), graduating summa cum laude in 1991.

Rhaban Hark received the master's degree in Electrical Engineering and the master's degree in 2015 at the Technical University of Darmstadt. Between 2015 and 2019, he was a Research Assistant with the Multimedia Communications Engineering Lab, Technical University of Darmstadt. In 2019, he received his Dr. Ing. at the Technical University of Darmstadt with the dissertation titled "Monitoring Federated Softwarized Networks: Approaches for Efficient and Collaborative Data Collection in Large-Scale Software-Defined Networks." Currently, he heads the Adaptive Overlay Communications Group at the Multimedia Communications Lab at the Technical University of Darmstadt and focuses on distributed monitoring approaches in the context of Software-Defined Networks.

Boris Koldehofe received the PhD degree from the Chalmers University of Technology, Gothenburg, Sweden, in 2005. He was a Senior Researcher and a Lecturer with the Technical University of Darmstadt and IPVS of the University of Stuttgart, Stuttgart, Germany. He is currently Professor of Computer Science at the University of Groningen and a Principal Investigator with the DFG Collaborative Research Center 1053 Multi-Mechanisms Adaptation for the Future Internet funded by the German Science Foundation. He has extensive research and teaching experience in the area of networked and distributed systems. In particular, he has focused on software-defined networks, adaptive communication middleware, and distributed event-based systems. He has contributed to more than 100 scientific publications in major journals and conferences. Prof. Koldehofe serves in several program committees and as a reviewer for high reputational journals.

Ralf Steinmetz is full professor at the Technical University of Darmstadt, Germany. In Darmstadt, he is the head of the Multimedia Communications lab as well as the Hessian Telemedia Technology Competence Center httc; see www.kom.tu-darmstadt.de Together with more than 30 researchers, he works towards his vision of "seamless adaptive multimedia communications." He has contributed to over 900 refereed publications; he is a Fellow of the IEEE, the ACM, and the VDE ITG.

How to cite this article: Kundel R, Nobach L, Blendin J, et al. OpenBNG: Central office network functions on programmable data plane hardware. *Int J Network Mgmt*. 2020;e2134. https://doi.org/10.1002/nem.2134