Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment

Leonhard Nobach*[1], Jeremias Blendin*[1], Hans-Jörg Kolbe†, Georg Schyguda†, David Hausheer*
*Peer-to-Peer Systems Engineering Lab, TU Darmstadt, Email: {lnobach,jblendin,hausheer}@ps.tu-darmstadt.de
†Deutsche Telekom AG, Darmstadt, Email: {Hans-Joerg.Kolbe,G.Schyguda}@telekom.de

***Abstract*—The current ecosystem of network elements, such as switches and appliances, is largely dominated by devices supplied and sold with a bundled operating system, and software dedicated to manage the device's forwarding hardware, however, these platforms are not open-source and cannot be arbitrarily customized, and there is no cost transparency or flexibility in choosing software different to the bundled components.**

In this paper, we explore the capabilities of bare-metal switches, which are equipped with commodity switching hardware components, but shipped without an operating system. We evaluate the feasibility of these commonly lower-cost devices to meet the requirements of a customized, carrier-grade network function. Therefore, we have implemented a prototype on generic hardware, re-using as much open-source software as possible. Our Broadband Remote Access Server (BRAS) prototype can lower the cost compared to proprietary network appliances, and, known to have a hardware backplane capacity of 720 Gbps, the merchant-silicon / ASIC approach can highly outperform the state of the art of current x86-based virtualized network functions, while implementing the most important BRAS features.

***Index Terms*—Bare-Metal Switching, Dataplanes, Network Functions, Middleboxes, Sofware-Defined Networking, Cost-Efficiency**

I. Introduction

Traditionally, a network switch is sold as an all-in-one device. It is guaranteed to provide a number of features, certain performance metrics, network, routing, spanning-tree and authentication protocols, as well as a (commonly vendor-specific) configuration interface. When a device is sold, a licensing contract is usually made with the same vendor to use the latter's software, specifically tied to the hardware of the switch.

When looking at the market of switches offered using this traditional business model, we can see that the number of vendors and their products is very large, however, the used hardware components are very similar. Even regarding the most essential components (like the forwarding ASIC[2]), a small group of hardware manufacturers supplies the switch vendors with the same products. In the last years, some switch vendors started with a new business model: The latter focuses on hardware manufacturing only, and sells switches made out of the aforementioned, well-known bulk components to a low price.

On the one hand, this approach has created a two-sided market, as new companies have emerged entirely focusing on *software* compatible with the switches, thus enabling the traditional and also some novel use cases. On the other hand, it opens up the opportunity for *customization* of the software to specific needs of the network operator, as, besides that network operators are challenged by the cost of traditional all-in-one forwarding hardware, they often have to adapt their internal processes to the heterogeneous software of the latter. To support free customization, an open-source community has emerged, providing tools and environments for software development [11].

In this work, we first motivate the scene for bare-metal switches (BMS) – which emerged from the market and open-source communities – as a new research area. Here, we also state commonalities and differences to software-defined networking. Secondly, we assess the feasibility and effort of creating customized software for a BMS under the requirements of large network operators, which procure networking hardware in high quantities, and thus have a larger opportunity to save license costs. Therefore, this paper has a special focus on *programmability* and *customization* of these switches from a technical perspective.

As a proof-of-concept, we have implemented and present in this paper a representative use case on a BMS using *publicly available* software and APIs only: a Broadband Remote Access Service (BRAS), which terminates home routers, provides them with an IP address, and handles their authentication and quality of service. Our qualitative test results verify successful implementation of all the *essential* features of a BRAS. Throughput performance results have been beyond our measurement capacities at the time of writing, however the device's capacity is given in the datasheet with 720 gigabits per second [6], [15].

The next section (Section II) provides the background of the BMS concept and introduces methods and APIs to program a BMS device. After discussing related work in Section III, we explain and reason the desired behavior (IV) and introduce our implementation (V) with a special focus on using the flow-based API to the ASIC (Section V-B). We finally present the evaluation results of our implementation in Section VI.

[1]since May 2017 with Multimedia Communications Lab (KOM), TU Darmstadt.

[2]Application-Specific Integrated Circuit

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

II. Background

In one sentence, a bare-metal switch, also known as white-box switch, is a **switch not bundled with an operating system (OS)**, which also means that the BMS vendors only provide warranty and support for the hardware. Furthermore, a BMS platform is targeted to be as open for OS customization as servers in a datacenter are. In all use cases we have seen, OS customization means to install a Linux-based operating system on the bare-metal switch (even Microsoft does that [14]).

The advantage of OS customization is that the switch operator can develop and configure the switch's control plane to the own needs. In a BMS-based infrastructure, there is no need to adapt the business processes and monitoring infrastructure to the interfaces provided by a switch vendor. Customized software components installed on servers (like monitoring and management) can even be re-used and installed on the switch hardware, as well.

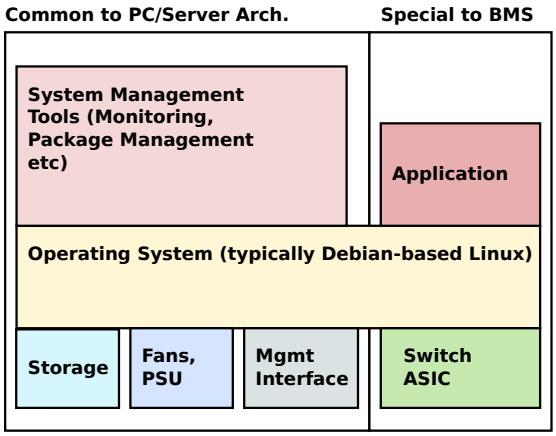

Fig. 1. BMS Generic Architecture

Figure 1 depicts a generic architecture of a bare-metal switch. At a first glance, the switch has an architecture which is very common to a commodity server, and from a management perspective, a bare-metal switch can be operated exactly as such: When installing a Linux-based OS (like OpenNetworkLinux) and connecting through the management interface, the switch will behave like a default Linux host[3]. From here, it is possible to install, compile and run Linux software as needed for own purposes.

The main difference of using a BMS compared to using a server is to get the *switchports* running, which is not possible with an out-of-the-box Linux so far. The switch ports will not be visible to Linux as network interfaces, this only applies to the management interface. Instead of appearing as individual devices to the Linux kernel, the switchports are tied together by a powerful *application-specific integrated circuit (ASIC)*, providing a single interface to the Linux kernel only. The operating system and the application commonly does not receive and send packets over this interface, instead it instructs the ASIC with rules on what to do with any packets coming in (e.g. to modify them), so called *forwarding rules*. The ASIC is capable of executing these rules on packets with a very high data rate compared to CPU-based I/O. Nevertheless, ASICs are also capable of accepting rules to directly forward packets to the control plane, or send packets from it, however this should be rarely used in order not to degrade performance.

To avoid the complexity of dataplane programmability, ASIC vendors have joined the BMS community and provide drivers, interfaces and abstractions to their dataplane [4]. OS vendors for BMS also provide alternatives to the low-level ASIC programming APIs with middlewares for true OpenFlow [13], Linux interfaces and bridges, or even a traditional command line, however with additional license costs.

A. BMS and the SDN concept

Bare-metal switching is actually the SDN concept driven very far and to a low level. Most software-defined network controllers are commonly relying on standardized, south-bound protocols like OpenFlow to tell switches what to do. However, SDN switches then translate commands of these standardized, south-bound protocol commands into changes in the ASIC forwarding tables (Forwarding Information Base (FIB) or Ternary Content-Addressable Memory (TCAM)).

Bare-Metal Switches generally do not support any standardized south-bound protocol out of the box, if such a protocol is desired, it must be implemented in the operating system first. An attempt to achieve this on a bare-metal switch is the publicly available OF-DPA (Section II-B) in combination with the Indigo OpenFlow agent. Although OF-DPA is standardized, the applied table restrictions to OpenFlow are very narrow, and likely will reject an OpenFlow control plane not specifically adapted to it. A proprietary alternative is PicOS [13], which aims for OVSDB and OpenFlow compatibility.

The idea of the BMS concept is that, instead of standardizing the network communication protocol between the controller and the switch, the local forwarding application programming interface (API) of the switch is standardized, and the SDN controller is extended to the switch CPU instead of being restricted to a remote server. This *distributed* concept has the potential to improve dataplane reaction times compared to delayed controller links (like when enabling forwarding upon a packet event), to reduce network traffic due to controller interaction, and to improve resilience of the control plane (if a controller or the link to it fails). Unfortunately, at the time of writing, there is no widely-supported vendor-neutral API to manage the ASIC, likely due to hardware differences.

B. OpenNSL and OF-DPA

As a major provider of switching ASICs, Broadcom has stepped up to the bare-metal switch community by supplying interfaces for ASIC programmability. The specification and libraries **OpenNSL** and **OF-DPA** are publicly available on GitHub [3]. They both consist of drivers, a daemon, and

[3]An exception is that the BM switches commonly do not have a VGA port and must be initially configured via the serial interface until SSH access is established.

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

interfaces. *OpenNSL* originally focuses on traditional network management commands. In particular, typical API actions in OpenNSL are about adding/removing VLANs and ports to VLANs, switching ports to L2/L3 mode, adding/removing routes, port mirroring, QoS, statistics, link aggregation, and VxLAN.

The *OpenFlow Dataplane Abstraction (OF-DPA)* is an API to modify the forwarding behavior of the ASIC in a flow table format. The operations like adding and removing flow tables, group actions, and meters, are very similar to OpenFlow, however the table types of its multi-table pipeline are very restricted in their functionality. Since June 2016, the OF-DPA APIs have become part of the OpenNSL library.

Like mentioned before, the similarity to OpenFlow allows the API to be directly accessed over this network protocol with an appropriate OpenFlow agent like *Indigo*. However, because of the very restricted set of possible matchers and actions in every table, the semantics of an OpenFlow controller must be adapted to the particular table type restrictions, which contradicts the idea of OpenFlow's hardware independence. The detailed OF-DPA specification is available on GitHub [4].

OF-DPA can be used either as an API by a local controller on the switch, or via OpenFlow by a remote controller. Regarding the first variant where we will focus on, a controller application can load a shared library, recommended for C and C++ development. For Python developers, a Python wrapper is also supplied with OF-DPA [5]. We have first tested the Python wrapper, which is stable and relatively easy to use, however, we experienced performance problems with controller packet-in/out operations, thus we have switched to the C header files.

To the best of our knowledge, there are no OF-DPA-based open-source applications at the time of writing. Although Python example scripts are given for a small set of operations, larger projects written in the C language could not be discovered.

C. The OF-DPA pipeline model

The OF-DPA pipeline [4] consists of two main entities, *tables* and *group actions*, which are both known from OpenFlow. Tables match a selected set of fields and can apply a selected set of actions on the packet and on several metadata fields, while group actions are a set of actions to be applied to a packet. Entries of several table types in OF-DPA can set a group action on a packet, which is only executed after the end of the ingress pipeline. This means a subsequent ingress table entry can also clear group actions previous tables have applied, which will lead to no effect of the group action. OF-DPA has very strict requirements on the sequence of tables which must be applied.

Although OF-DPA comprises a large number of table and group action types (some of them are for MPLS termination/initiation and for VXLAN support), most VLAN-based L2/L3 use cases only require a small excerpt of it. This is caused by the fact that most tables have a built-in default action which forwards a packet to the next relevant table. In this document, we therefore restrict our explanation to the likely most required ones: the **VLAN**, the **Policy ACL** flow table, the *L2 Rewrite* and the *L2 Interface* group action. With these elements alone, it is possible to implement a Layer 3 hop with multi-field flow matching.

- The **VLAN** table matches the input port and the (first) VLAN tag only. If no entry exists in the VLAN table, a packet is dropped, which constitutes a VLAN filter per port. Except in special cases like removing a second VLAN tag, VXLAN or MPLS L2 initiation, the successor of the VLAN table is the Termination MAC flow table.
- The **Policy ACL** table can be seen as most powerful one: It supports wide-field *matching* on most packet headers, comparable to current OpenFlow versions. It is also possible to apply meters here. However, instead of applying versatile actions as in OpenFlow, the table is restricted to applying only the following *group actions*.
- An **L2 Rewrite group action** is applied by the Policy ACL table. It can rewrite the *VLAN ID* and the source and destination *MAC addresses* (note that IP rewriting is not possible in the current OF-DPA version). The L2 Rewrite group action must apply an L2 Interface group action afterwards.
- The **L2 Interface** group action can be applied either directly or via the L2 Rewrite group action. It is just defined as a tuple consisting of an *output interface* and *output VLAN*. When applied, the packet will be sent out on the respective port tagged with the given VLAN. If the packet's VLAN does not match the one in the action, the packet is dropped (therefore, the L2 Interface group action is a kind of *VLAN filter*).

D. Broadband Remote Access Servers

A broadband remote access server (BRAS) is a fundamental network function in an ISP's network. The function terminates DSL access multiplexers (DSLAMs) from the subscriber side and provides access to the IP network. In some networks, the BRAS uses PPP over Ethernet (PPPoE) at the subscriber's side to terminate multiple subscribers, in other networks, VLAN-based subscriber termination is used where we will focus on. A BRAS has strong requirements on performance, as it terminates and services a very large number of subscribers. Therefore, the BRAS network function is commonly implemented in appliances with special ASIC support, which can be assumed to be very expensive compared to bare-metal switches.

III. Related Work

Three years ago, first contributors started to emancipate from purchasing tightly-coupled hardware-software-systems. Int 2014, Facebook announced the Open Switching System (FBOSS) at the Open Compute Project (OCP) Engineering Summit [7]. The announcement describes a hardware switching platform consisting of widely available commercial-off-the-shelf (COTS) components.

While especially OpenFlow-based SDN switches have widely been subject to research [10], investigating the oppor-

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

tunities and drawbacks of bare-metal switches is still a very new topic for the research community. Kurtz et al. [9] have compared a virtual switch (an Open vSwitch instance) and a PicOS-based bare-metal switch regarding performance in the context of critical infrastructures. While the bare-metal switches outperform virtualized instances especially in forwarding latency, failovers have been found as slower regarding the former.

Efforts to implement the functionality of a BRAS in an SDN/NFV environment have also been recently proposed in the *Central Office Re-designed as a Data Center (CORD)* architecture [12]. CORD envisions to replace the specialized hardware currently used to terminate subscribers, including an optical line termination (OLT), a BRAS, and other related components, with commodity hardware servers. CORD also comprises bare-metal switches, however dedicated for Layer 2 or MPLS packet forwarding between VNF instances only. The major difference to our approach is that, in CORD, the BRAS runs in a VNF (as a virtual subscriber gateway), instead of the bare-metal switch's ASIC used in our prototype.

Several other works have adopted the OF-DPA abstraction for their use case [8], [16], [17], including an SDN testbed and an OF-DPA controller.

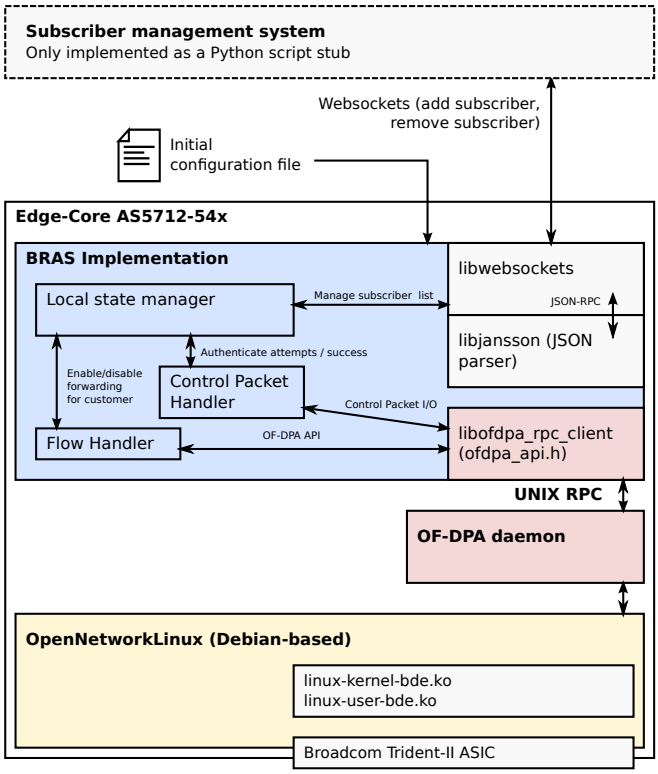

Fig. 2. System architecture of the BRAS implementation

IV. System Design

We will only briefly specify the desired behavior in this section. The BRAS has two internally-configured MAC addresses, the *subscriber-facing* and the *core-facing* one. Several ports are being designated for termination of the subscriber's side, the Optical Line Termination (OLT) ports. Note that their physical layer can be easily adapted by exchanging the SFP module, e.g. with a module with GPON[4] support or a special wavelength. Besides the subscriber-facing ports, **core** ports can be configured on the switch. These ports support a simple IP/Ethernet format and thus can be attached to core networks.

In the upstream direction, the dataplane must detect and redirect subscriber authentication attempts and control packets to the CPU. The BRAS must furthermore ensure that only authenticated subscribers (identified by their VLAN ID) can send packets originating from their designated IP addresses, obtained from a preconfigured lease pool. A three-color-based *meter* must be applied[5] to ensure subscribers can only consume the bandwidth of the purchased service. In the downstream direction, the BRAS must look up the packet's destination IP address, and if it belongs to an authenticated subscriber, forward it to the subscriber's VLAN ID.

Additionally, a subscriber is assigned to a set of services (like Internet, VoIP and IPTV), which are either specified by an additional inner VLAN tag, or by the IP subnet which is used. Upon authentication, the BRAS terminates all services the subscriber is assigned to, and forwards it to a specified core port or VLAN.

V. Implementation

Our implementation of the aforementioned BRAS is planned to be realized with as much free software as possible. Therefore, we have selected OpenNetworkLinux (ONL) as the operating system platform (Figure 2). For performance reasons, the BRAS control plane is written entirely in the C language (depicted in blue). However, proprietary components cannot be avoided, as we require them for ASIC support. These proprietary components include the OF-DPA binary and the OF-DPA RPC client (depicted in red). As the BRAS control plane runs locally, we have decided to use OF-DPA without an OpenFlow agent and directly link against the OF-DPA API.

The BRAS software is loaded with an initial configuration file (see Figure 3 for an example). The values in this configuration file are related to the supported services, subnets, core ports and MAC addresses, and cannot be changed at runtime. Upon startup, initial flow rules are installed (Section V-B) and a Websockets server is started to accept subscriber configuration at runtime.

Figure 4 depicts the subscriber state model. Initially, a subscriber is not existing (white) in the device configuration. If a subscriber is added to the device via the Websockets connection, the subscriber is *initialized* (red): The *local state manager* installs flow rules which only forward authentication packets from the subscriber to be passed to the controller, however no

[4]Gigabit Passive Optical Network
[5]RFC 2697, 2698 or 4115

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

```
{
  "configuration": {
    "global_info": {
      "_service_list": [
        {
          "_srv_name":"Internet",
          "_srv_subnet":"100.50.1.0/24",
          "_srv_core_port":"7",
          "_srv_vlan_tag":"4",
          "_srv_next_hop_mac":"00:1b:21:1c:e7:4b",
          "_srv_untagged_flag":"1",
          "_srv_gateway_ipv4_addr":"100.50.1.254",
          "_srv_gateway_ipv4_mask":"255.255.255.0"
        }, {
          "_srv_name":"IPTV",
          "_srv_subnet":"192.168.2.0/24",
          "_srv_core_port":"8",
          "_srv_vlan_tag":"8",
          "_srv_next_hop_mac":"00:30:48:8a:cd:d0",
          "_srv_untagged_flag":"0",
          "_srv_gateway_ipv4_addr":"192.168.2.254",
          "_srv_gateway_ipv4_mask":"255.255.255.0"
        } ],
      "_ipv6_prefix_length":"64",
      "_bras_sub_mac":"52:54:00:00:00:01",
      "_bras_core_mac":"52:54:00:00:00:02"
    }
  }
}
```

Fig. 3. Example initial JSON configuration file, defining three services: Internet (VLAN 4) and IPTV (VLAN 8). The internet service uses the core port 7 (untagged), and IPTV uses the core port 8 (tagged). Subscribers and their ports/VLANs are configured at runtime.

forwarding is yet enabled. The *Control Packet Handler* then waits for authentication attempts of the subscriber, and if the authentication has been successful, the forwarding state for all the subscriber's services to the core network and back is established, the subscriber is then *authenticated* (blue). In the other direction, a subscriber becomes deauthenticated by not refreshing the authentication state in a pre-defined interval. Finally, a subscriber can be removed from the local state (e.g. by ending the service contract or moving to another BRAS region).

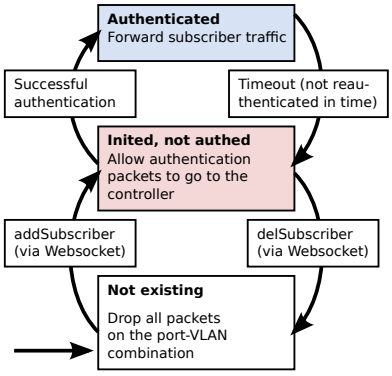

Fig. 4. Per-subscriber state diagram in the BRAS switch

A. Subscriber Configuration and Authentication

To initialize subscribers, a Websocket service is started which is accessible on the BRAS management port; subscribers can be added and removed by connecting to this

```
{"_cmd":"adduser","_arg":{
        "_subsc_id":"1002",
        "_username":"john_doe",
        "_password":"12345",
        "_port_num":"5",
        "_vlan_id":"11",
        "_service_tags":[ "4", "8"]
}}
```

Fig. 5. Example runtime configuration of a subscriber. If successful, the command is answered with 'OK', with an explanatory failure notice otherwise.

service. Figure 5 shows how a subscriber with a specified username and password is added in a JSON format. The specified OLT port and the VLAN ID is expected when the subscriber authenticates. Furthermore, the subscriber is restricted to the given service VLANs. The subscriber ID is only for internal use (e.g. to be able to delete the subscriber when needed) and can be arbitrarily chosen.

In a BRAS for carrier-grade productive usage, standard formats for authentication should be followed. A candidate for a state-of-the-art standardized authentication protocol is EAP over LAN (EAPOL), which is part of the IEEE 802.1X authentication standard. Subscribers requesting authentication can use this protocol via a Ethernet-based message exchange in conjunction with a RADIUS server in the provider's backend. However, authentication is a task conducted by the control plane, the dataplane's task is only to forward authentication packets to the controller (or the RADIUS server). The implementation of an EAPOL support is hard, to simplify the PoC implementation targeted to assess ASIC programmability for the dataplane, we have designed a simple, light-weight authentication protocol. It is plain-text username-password-based, and therefore not secure especially against eavesdropping. We argue that to extend the PoC to EAPOL support, only the C-based control plane code must be extended, but not the dataplane capabilities which we want to evaluate.

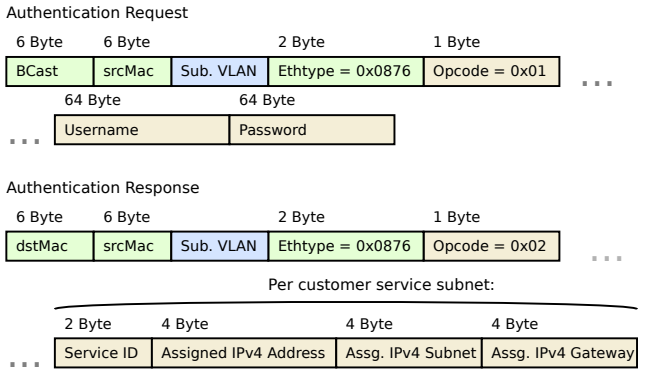

Fig. 6. BRAS Authentication Packet Format

Figure 6 shows the packet format for the authentication request and response. Authentication requests (initially sent by the client) are always using the broadcast address on the Ethernet layer, as the destination MAC address is assumed to be previously unknown. The source MAC address is very

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

important, as, upon successful authentication, it is used as the future CPE WAN interface address, the only address allowed to send packets through the BRAS dataplane and the target MAC address for any packets sent to the subscriber. Next, the subscriber's VLAN is expected (assumed to be assigned by the DSLAM, which is agnostic of any authentication). The ethertype of the next header of the VLAN tag is set to `0x0876` which we have chosen as the ethertype of the custom protocol. After the opcode (`0x01`), the username and password fields are both of a 64-byte fixed size, containing null-terminated strings.

In the response packet, the destination MAC address contains the CPE which made the request, and the interior BRAS MAC address as the source, and the requesting subscriber's VLAN. The ethertype is equal to the request, but the opcode of the response is `0x02` upon authentication success, `0x03` otherwise. Upon success, for every service available, the service VLAN ID (2 byte), the assigned IPv4 address, the subnet mask, and the default gateway are provided[6].

B. Flow Model

In this section, we describe the dataplane implementation in OF-DPA 2.01. To a certain extent, the dataplane follows the behavior of a typical Layer 3 hop (a router), which is straightforward to implement in OF-DPA. The greatest challenges however for a scalable implementation in OF-DPA are beyond these functionalities, like the *antispoofing*, which requires a large table for source IP matching, as well as a fine-grained *metering* support. In the following, we describe two dataplane modes, the *double-tagged* and the *single-tagged* mode. The latter mode was introduced, as the implementation of the double-tagged mode was not supported by the present OF-DPA API version.

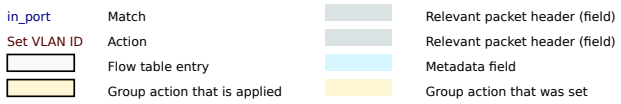

Fig. 7. Meaning of the elements used in the following OF-DPA flow model figures.

Figure 8 depicts the flow model we have used for the double-tagged mode in the packet flow direction from the subscriber to the core (see Figure 7 for the meaning of the symbols). The L2 Rewrite and L2 Interface group actions have to be instantiated only once per service in this direction, the VLAN table entry must be instantiated per subscriber, while a VLAN 1 and the Policy ACL entry must be instantiated for every subscriber times every service. The handling of the ingress double tag was supported and successfully tested in this direction. In the pipeline, we introduce the term *ASIC VLAN* as an only internally-used VLAN ID, carrying information which would otherwise be lost between the tables due to the stripping of the outer VLAN tag. The ID can be arbitrarily chosen, however must be unique on the ingress port to be able to uniquely match the packet in the subsequent Policy ACL table. As the number of different VLAN IDs is 4096 (or a little bit less), the maximum number of subscribers times services on every port is restricted to this number.

Figure 9 shows the double-tagging mode in the direction from the core to the subscriber. Despite the support of a large number of entries to match the destination IP address in the Unicast Routing flow table, we had to use the Policy ACL table to be able to apply per-subscriber metering. The pipeline described here is not valid, as an Egress VLAN table entry, which is required for the Egress VLAN 1 table to add a second tag, cannot follow an L2 Interface group action. The Egress VLAN table is only accepted after an L2 Unfiltered Interface group action (here, the metadata field `ALLOW_VLAN_TRANSLATION` is set to 1), which itself can only be used in MPLS intiation or termination use cases.

Figure 10 shows the flow model for the *single-tagged* mode in the packet flow direction from the subscriber to the core. The only difference is that no service VLAN tag is used, instead, the service is identified by the source IP subnet of the subscriber. This approach is functional, provides a comparable isolation, and does not impose the scalability restrictions of the ASIC VLAN approach, leading to only 4096 subscribers times services.

In the packet flow direction of the *single-tagged* mode from the core to the subscriber (Figure 11), no egress tables are required, thus, the pipeline of this mode is functional. Like in the double-tagged mode, the Policy ACL table must be used to be able to apply a per-subscriber meter, the large Unicast Routing table cannot be exploited.

VI. Evaluation Results

Figure 13 depicts the test setup we have used to evaluate correct operation of the BRAS PoC. Therefore, we have connected a test server to the switch with four Gigabit Ethernet ports: `dslam0` (Switchport 5), `dslam1` (Switchport 6), `core0` (Switchport 7), `core1` (Switchport 8). On the test server, we have implemented two test customer premises devices (CPE) in LXC containers (subsc1 and subsc2), every traffic from or to these containers has been tagged with an individual VLAN ID (resembling the behavior of DSLAMs) on `dslam0`. In various tests, we forwarded tagged traffic to `dslam1` instead, to verify correct behavior with multiple ports. The CPE LXC containers use a Python script as the authentication client at the BRAS.

The interface `core0` has been split into two subinterfaces for the service VLANs 8 and 10, which have been configured as tagged core interfaces in the BRAS, `core1` has been configured as the service VLAN 4, but untagged.

The connectivity between the subscriber VMs and the core networks was tested with ping and iperf3.

We have focused on functional tests and table size restrictions, as we have assumed these to be the primary bottlenecks of the platform. Table I shows the implementation and testing

[6]We implemented the IPv6 dataplane, but we did not implement IPv6 control plane functionality.

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

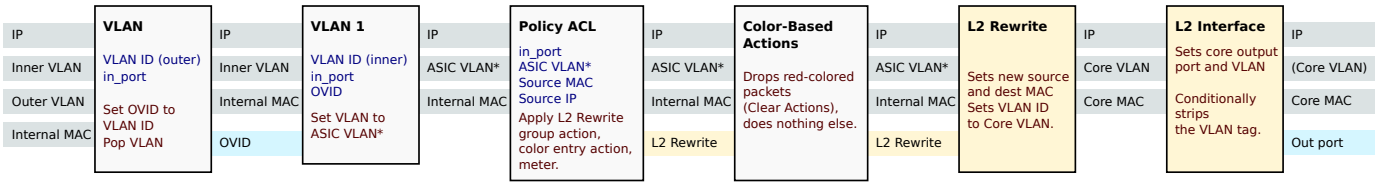

Fig. 8. OF-DPA flow model of the double-tagged mode, upstream direction

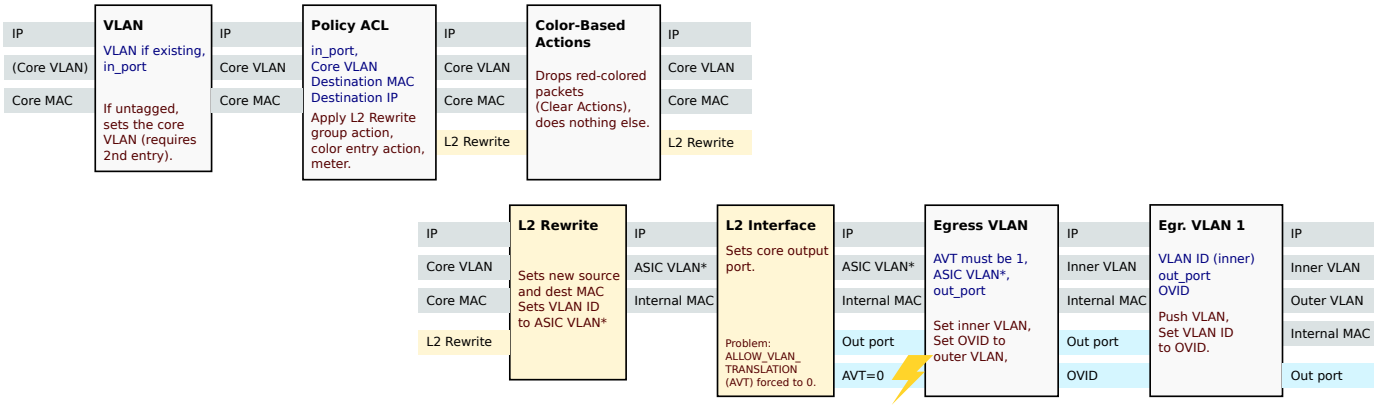

Fig. 9. OF-DPA flow model of the double-tagged mode, downstream direction

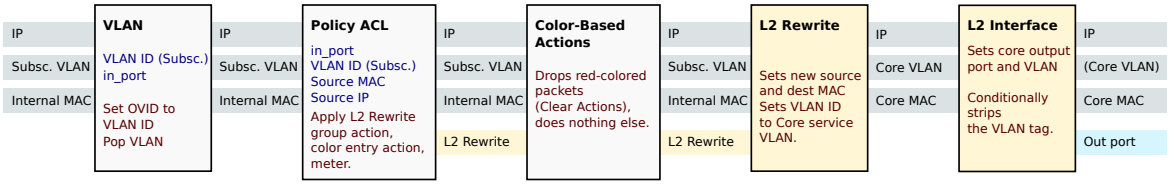

Fig. 10. OF-DPA flow model of the single-tagged mode, upstream direction

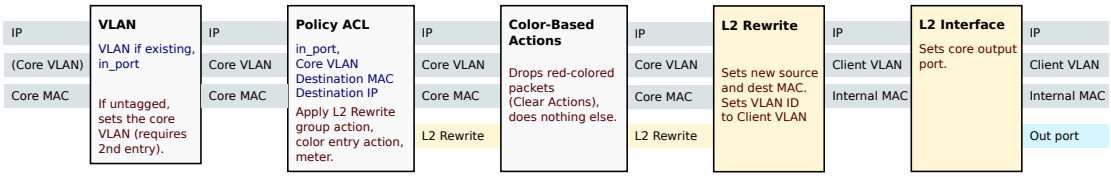

Fig. 11. OF-DPA flow model of the single-tagged mode, downstream direction

Feature	OF-DPA Support	Impl. Status	Test Status	Essential Feature
VLAN Single Tagging	Yes	Done	Success	Yes
VLAN Double Tagging	No	-	-	No
Antispoof	Yes	Done	Success	Yes
IPv6	Yes	DP-only	Success	Yes
Metering (RFC 2697 ff.)	Yes	Done	Not tested	Yes
Authentication	-	Done	Success	Yes
Liveness	-	Done	Success	No
ARP Responder	-	Done	Success	Yes
Lease from IP Pool	-	Done	Success	Yes

TABLE I
BRAS IMPLEMENTATION AND TESTING STATUS

status of several features of our BRAS PoC, we also provide whether the features are *essential* for a BRAS implementation or not: In this context, essential features are the ones that are required to build a BRAS. This e.g. applies to authentication, provisioning of leases, and ARP responding. However, VLAN double-tagging (a.k.a. Q-in-Q or IEE 802.1ad) can be used to increase the identifier space of VLANs, or to apply hierarchical pushing/popping of the VLANs, and is desired in carrier networks, however, it is possible to implement a functional BRAS without it.

In our prototype, we could successfully implement and test all these essential features of a BRAS. As already mentioned,

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

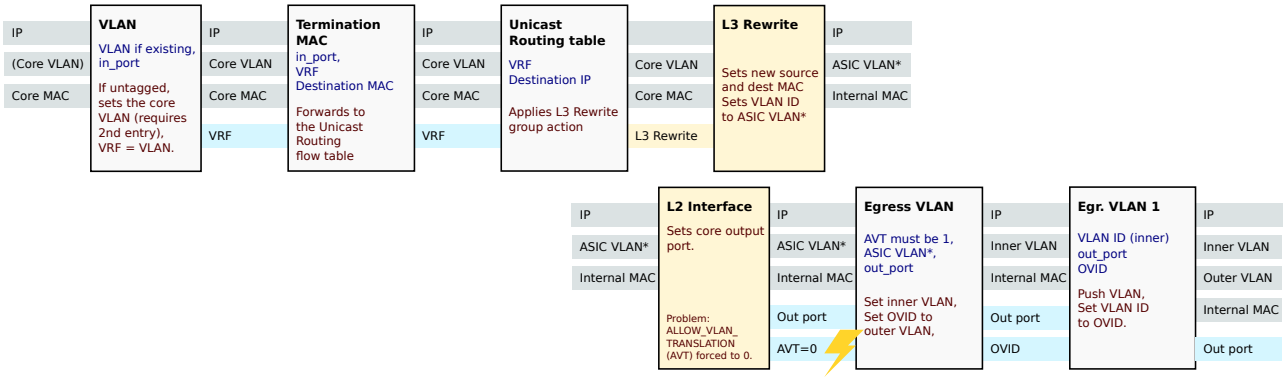

Fig. 12. OF-DPA flow model of the single-tagged mode when using the Unicast Routing table, downstream direction

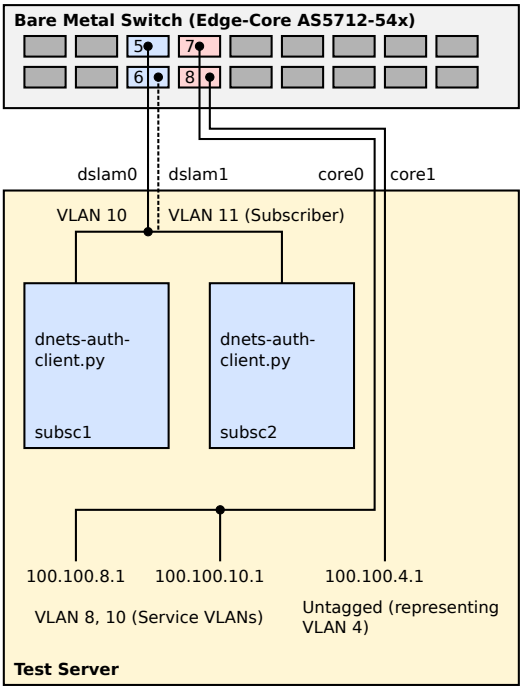

Fig. 13. BRAS PoC test setup.

we have not been able to successfully implement a functional double-tagging in the OF-DPA pipeline. At the time of writing, it was not available in the published programming interface [4]. Due to the extraordinarily high backplane capacity of the switch according to the hardware descriptions (720 gigabits per second), we could not verify an upper limit for throughput caused by the limitation of our testing equipment. Current limitations in the evaluation hardware (load generators) also did not allow to test the meters.

Another aspect is *scalability*. We identified the Policy ACL flow table to be the primary bottleneck to create a scalable implementation on the AS5712-54x (Trident II), which could manage up to 3072 entries. According to our flow model, the switch requires two flows per subscriber times subnet, in a dual-stack implementation even 4 entries are required. Even if a dual-stack subscriber uses 1 service VLAN, 768 subscribers are supported per switch only. Using the Unicast Routing table for the downstream direction doubled the number of possible subscribers, however a metering support is not intended by this table. Nevertheless, Broadcom and other vendors have announced new chipset generations with support for much larger flow tables [15]. Thus, the current scalability limits are not expected to persist.

A. Next Steps

While we have conducted functional tests of the BRAS implementation, the PoC still needs to undergo stress and performance tests in a next step, which requires several additional high-capacity servers and specialized hardware to cope with the high backplane capacity of the hardware. Recommended tests include setting up, adding and removing large numbers of subscribers, or conducting parallel high-throughput and latency tests on multiple ports. Furthermore, configuration input - either in the initial config file or the Websocket-based runtime configuration - must be validated. In the current state, for example, it is possible to configure multiple service VLANs on a single physical port as untagged. Finally, especially because of the scalability restrictions we experienced with the studied silicon, we will also evaluate chipsets from the upcoming new generations.

VII. Conclusion and Outlook

The bare-metal switch hardware market is well-established, and open-source projects are becoming more and more mature to support switch customization. The popularity of Broadcom-based chipsets highly influences the possibilities of unique *programmability*, thus OF-DPA can currently be seen as the most prominent open interface to program a bare-metal switch pipeline on a flow level. For simple use cases up to Layer 3 routing, and for the essential requirements on a BRAS, OF-DPA has been found as suitable and well-usable. With increasing hardware support since January 2017, vendor-independent API candidates like the OCP's Switch Abstraction Interface

Leonhard Nobach, Jeremias Blendin, Hans-Jörg Kolbe, Georg Schyguda, David Hausheer:
Bare-Metal Switches And Their Customization And Usability In A Carrier-Grade Environment.
In: Proc. 42st IEEE Conference on Local Computer Networks (LCN), pp. 649-657, IEEE, October 2017.
ISBN: 978-1-5090-6523-3

(SAI) might gain more and more relevance in the near future, at least for *traditional* switching and routing use cases.

With a publicly available programming interface of a commoditized chipset, we have been able to implement the BRAS regarding the essential features, basically providing the proof that it is rather a matter of evolution of existing merchant silicon and programming interfaces to enable network operators to shift to this technology. Now, it is desired to also support advanced BRAS features like point-to-point protocol (PPPoE) termination. We are looking forward to upcoming generations of switching chipsets: Future hardware enables protocol-independent pipeline programming and customization with a terabit or multi-terabit backplane capacity [1], but – as opposed to ASICs – specified in software by using open languages like P4 [2]. Thus, the need for an ASIC may not be present anymore even for highest-performance network functions, and it will finally – as envisaged when SDN came up – only be a matter of software, whether a switch becomes a BRAS, a traffic shaper, or a firewall. Our efforts for prototyping displayed clearly that also on the software side, a major change is underway. Especially the progress of open switch abstraction layer definitions will play a key role in further separating hardware from software.

Acknowledgements

This work has been supported by Deutsche Telekom through the *Dynamic Networks 5* project. Furthermore, we thank our colleagues and the reviewers for their valuable input and feedback.

References

[1] Barefoot Networks. Tofino, May 2017. https://www.barefootnetworks.com/technology.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *SIGCOMM CCR*, 44(3):87–95, 2014.

[3] Broadcom. Broadcom-Switch on GitHub, December 2016. https://github.com/Broadcom-Switch/.

[4] Broadcom. OpenFlow Data Plane Abstraction (OF-DPA): Abstract Switch Specification, Version 2.01, December 2016. https://github.com/Broadcom-Switch/of-dpa/blob/master/OFDPAS-ETP100-R.pdf.

[5] Broadcom. OpenFlow Data Plane Abstraction (OF-DPA): Python wrapper, December 2016. https://github.com/Broadcom-Switch/of-dpa/blob/master/bin/as6700-trident2-fsl14/OFDPA_python.py.

[6] Edge-Core. AS5712-54x Product Information, May 2017. http://www.edge-core.com/productsInfo.php?cls=1&cls2=8&cls3=44&id=15.

[7] Facebook. Facebook Open Switching System, November 2013. https://code.facebook.com/posts/681382905244727.

[8] D. Fritzsche, Z. Magyari, M. Schlosser, and T. Jungel. Basebox – integrating whitebox switches into linux: A controller implementation for OF-DPA hardware. In *EWSDN*, pages 52–54. IEEE, 2016.

[9] F. Kurtz, N. Dorsch, and C. Wietfeld. Empirical comparison of virtualized and bare-metal switching for SDN-based 5G communication in critical infrastructures. In *NetSoft*, pages 453–458, June 2016.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, Mar. 2008.

[11] Open Network Linux. Open Network Linux, December 2016. https://opennetlinux.org.

[12] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow. Central office re-architected as a data center. *IEEE Communications*, 54(10):96–101, October 2016.

[13] Pica8. PicOS datasheet, June 2016. http://www.pica8.com/wp-content/uploads/2015/09/pica8-datasheet-picos.pdf.

[14] K. Subramaniam. Microsoft showcases the Azure Cloud Switch (ACS), December 2015. https://azure.microsoft.com/en-us/blog/microsoft-showcases-the-azure-cloud-switch-acs/.

[15] R. Toghraee. Comparing the Broadcom silicons used in datacenter switches, June 2016. https://www.linkedin.com/pulse/comparing-broadcom-silicons-used-datacenter-switches-reza-toghraee.

[16] S.-Y. Wang and I.-Y. Lee. Minireal: A real SDN network testbed built over an SDN bare metal commodity switch. In *ICC*, pages 1–6. IEEE, 2017.

[17] S.-Y. Wang, S.-Y. Liu, and C.-L. Chou. Design, implementation and performance evaluation of software openflow flow counters in a bare metal commodity switch. In *Computers and Communication (ISCC), 2016 IEEE Symposium on*, pages 651–656. IEEE, 2016.