The Road to BOFUSS: The Basic OpenFlow Userspace Software Switch

Eder Leão Fernandes

Queen Mary University of London, UK

Elisa Rojas and Joaquin Alvarez-Horcajo

University of Alcala, Spain

Zoltàn Lajos Kis

Ericsson, Hungary

Davide Sanvito

Politecnico di Milano, Italy

Nicola Bonelli

University of Pisa, Italy

Carmelo Cascone

Open Networking Foundation, USA

Christian Esteve Rothenberg

INTRIG, University of Campinas (UNICAMP), Brazil

Abstract

Software switches are pivotal in the Software-Defined Networking (SDN) paradigm, particularly in the early phases of development, deployment and testing. Currently, the most popular one is Open vSwitch (OVS), leveraged in many production-based environments. However, due to its kernel-based nature, OVS is typically complex to modify when additional features or adaptation is required. To this regard, a simpler user-space is key to perform these modifications.

In this article, we present a rich overview of BOFUSS, the basic OpenFlow user-space software switch. BOFUSS has been widely used in the research community for diverse reasons, but it lacked a proper reference document. For

this purpose, we describe the switch, its history, architecture, uses cases and evaluation, together with a survey of works that leverage this switch. The main goal is to provide a comprehensive overview of the switch and its characteristics. Although the original BOFUSS is not expected to surpass the high performance of OVS, it is a useful complementary artifact that provides some OpenFlow features missing in OVS and it can be easily modified for extended functionality. Moreover, enhancements provided by the BEBA project brought the performance from BOFUSS close to OVS. In any case, this paper sheds light to researchers looking for the trade-offs between performance and customization of BOFUSS.

Keywords: Software-Defined Networking, Software switches, OpenFlow, Open source, Data plane programmability

---

1. Introduction

Over the last decade, Software-Defined Networking (SDN) has been enthroned as one of the most groundbreaking paradigms in communication networks by introducing radical transformations on how networks are designed, implemented, and operated [1]. At its foundations, SDN data plane devices (aka. switches) are featured with programmable interfaces (e.g., OpenFlow [2]) exposed to controller platforms. More specifically, open source software switches are a pivotal piece in the initial phases of research and prototyping founded on SDN principles.

Due to their wide use, two open source OpenFlow software switches deserve special attention: Open vSwitch (OVS) [3] and Basic OpenFlow User Space Switch (BOFUSS) [4]. Both have different characteristics that make them the best choice for different types of scenarios, research and deployment objectives. OVS is probably the most well-known SDN switch and used in commercial environments, mostly in SDN-based datacenter networks based on micro-segmentation following an overlay model (cf. [1]). BOFUSS is commonly seen as a secondary piece of software switch, mostly used for research purposes,

Proof-of-Concept (PoC) implementations, interoperability tests, among other non-production scenarios.

In this article, we present the history of BOFUSS going through a comprehensive overview of its architecture, applications, and evaluation. Let us start the journey by clarifying that BOFUSS is the name we have chosen for this "late baptism", since the switch did not have consistently used official name. Many authors denominate it as *CPqD switch*, being CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) the research and development center located in Campinas, Brazil, where it was developed, funded by the Ericsson Innovation Center in Brazil. Hence, the switch has been also referred to as *CPqD/Ericsson switch*, not only for the funding but also for the original code base from an OpenFlow 1.1 version developed by Ericsson Research TrafficLab [5] after forking Stanford OpenFlow 1.0 reference switch/controller implementation [6] developed around 10 years ago. *OF13SS* (from OpenFlow 1.3 software switch), or simply *ofsoftswitch13* (following its code name in the GitHub repository [7]), add to the list of names the software artefact is referred to. We believe this naming issues can be explained by the lack of an official publication, since the only publication focused on the tool [4], written in Portuguese, did not introduce a proper name and mainly used the term *OpenFlow version 1.3 software switch*.

Fixing our historical mistake of not having given a proper name (i.e. BOFUSS) to the widely used switch is one of the target contributions of this article. We delve into the switch history and architecture design in Section 2. Next, Section 3 presents selected use cases, which are later expanded in Section 4 through an extensive survey of the works (35+) that leverage BOFUSS in their research production. We evaluate and benchmark BOFUSS in Section 5 and, finally, we conclude the article in Section 6.

2. BOFUSS: Basic OpenFlow Userspace Software Switch

This section first introduces the history and motivation behind the development of BOFUSS, and then presents its design and architecture.

2.1. Brief History

Up until the release of the OpenFlow 1.0 standard, there were three OpenFlow switch implementations that provided more or less full compliance with the standard: i) The Stanford Reference OpenFlow Switch [6], which was developed along with the standardization process and its purpose was to provide a reference to OpenFlow switch behavior under various conditions; ii) The OpenFlow Python Switch (OFPS), which was implemented as part of the OFTest conformance testing framework [8], meant primarily as a testing framework, and iii) OVS [3, 9], the most popular and high performance virtual switch with OpenFlow support.

Since the beginning, the OpenFlow standardization process requires that all proposed features are implemented before they are accepted as part of the standard. During the OpenFlow 1.1 standardization work, most of the new feature prototypes were based on OVS, mostly on separate branches, independent of each other. Unfortunately, standardization only required that each individual new feature worked, instead of looking for a complete and unique implementation of all features, as a continuous evolution of the standard and SDN switches. As a result, when OpenFlow 1.1 was published, no implementation was available. While the independent features were implemented, they applied mutually incompatible changes to the core of the OVS code, so it was nearly impossible to converge them into a consistent codebase for OVS with complete support for OpenFlow 1.1.

This lead to the development of BOFUSS, as already explained in the introduction, popularly known as *CPqD* or ofsoftswitch13 among other code names. The core idea was the need of a simpler implementation to be used for multiple purposes such as: i) a reference implementation to verify standard behavior,

ii) an implementation with enough performance for test and prototype deployments, and iii) an elementary base to implement new features with ease.

The code of the switch was based on the framework and tools provided by the Reference OpenFlow Switch. Nevertheless, the datapath was rewritten from scratch to make sure it faithfully represented the concepts of the OpenFlow 1.1 standard. Additionally, the OpenFlow protocol handling was factored into a separate library, which allowed, for example, the implementation of the OpenFlow 1.1 protocol for the NOX controller. The first version of this switch was released in May 2011 [10].

Afterwards, the software became the first virtual switch to feature a complete implementation of OpenFlow 1.2 and 1.3, showcasing IPv6 support using the OpenFlow Extensible Match (OXM) syntax [11]. Because of the comprehensive support to OpenFlow features and the simple code base, the switch gradually gained popularity both in academia and in open-source OpenFlow prototyping at the Open Networking Foundation (ONF).

2.2. Design and Architecture

In order to understand the features of BOFUSS, the following sections describe its design and architecture. Furthermore, an architectural comparison with OVS (its direct competitor) is also performed to clarify the main differences.

2.2.1. Design choices and programming models

Design. The design and implementation of software for virtual switches are intricate work, requiring developers' knowledge of low-level networking details. Even though it is hard to escape the complex nature of software switches, BOFUSS main design goal is simplicity. Therefore, its implementation seeks for ease in understanding and modifying the OpenFlow switch. OpenFlow does not specify data structures and algorithms to support the pipeline of the protocol. It gives freedom to virtual switch designers to choose the structure of components to realize the pipeline described by the specifications. In the design

of BOFUSS, whenever possible, the implementation of the OpenFlow pipeline follows the most straightforward solutions. Frequently, the most direct approach is not the most efficient, but exchanging performance for simplicity is a trade-off worth paying, especially when prioritizing fast prototyping in support of research. We describe core design decisions that make BOFUSS an accessible option for faster prototyping.

Event handling. BOFUSS processes packets and OpenFlow messages in a single-threaded polling loop. First, it goes through the list of ports looking for received packets ready for processing. The pipeline then processes the sequence of available packets. Next, the switch iterates through remote connections with OpenFlow controllers to handle OpenFlow messages. Finally, when no more tasks are available, the switch blocks until a new event is available or for a maximum of 100ms. This pattern is typical of event-driven applications, also found in production-ready solutions such as Open vSwitch (OvS). However, OvS leverages multiple threads to speed up the setup of flows, cache revalidation, and polling statistics. While multiple threads improve performance, they also add extra complexity to the userspace code. BOFFUS single-threaded nature reflects the option for a more uncomplicated implementation instead of performance.

Packet Parsing. Another choice in BOFUSS that exchanges performance for more convenient addition of new protocols to OpenFlow is on the design of the packet parser. Because of the increase in the number of fields from OpenFlow 1.0 (14 fields) to 1.3 (40 fields), we realized the need for a solution that allows faster prototyping in fields. Our approach leverages NetPDL [12] to define the fields supported by the switch and integrates the Netbee library [13] to automate the parsing of fields. The following code listing shows an example of the definition of the UDP protocol in NetPDL. The children of the `fields` element must be in the order of the protocol header. Furthermore, we use the `longname` field to encode the values for the vendor's class and the field number according to the OpenFlow specification. This addition is essential to set the packet's matching fields correctly.

```
<?xml version="1.0" encoding="utf-8"?>
<netpdl name="nbee.org NetPDL Database" version="0.9"
    creator="nbee.org" date="09-04-2007">
<protocol name="udp" longname="UDP (User Datagram protocol)"
    showsumtemplate="udp">
    <format>
        <fields>
            <field type="fixed" name="sport" longname="{0x8000 15}"
             size="2" showtemplate="FieldDec"/>
             <field type="fixed" name="dport" longname="{0x8000 16}"
             size="2" showtemplate="FieldDec"/>
             <field type="fixed" name="len" longname="Payload length"
             size="2" showtemplate="FieldDec"/>
             <field type="fixed" name="crc" longname="Checksum"
             size="2" showtemplate="FieldHex"/>
        </fields>
    </format>
</protocol>
</netpdl>
```

The extensible nature of BOFUSS's packet parser showed efficiency in handling tricky fields to parse, such as the IPv6 Extension Header [14]. However, the addition of the Netbee module decreased the performance of the switch by a factor of three times. Researchers looking for better performance instead of easiness to extend the OpenFlow fields can easily replace the parser with their implementation. The change requires only modifying a single function of the packet handler that serves as the programming interface for the parser of the switch. This scenario is one example of how BOFUSS provides a useful base for modifying or prototyping new OpenFlow functionalities.

OpenFlow version. In the design of BOFUSS, we decided to support only a single version of the protocol. The virtual switch supports OpenFlow 1.3.5, the last version of the long-term branch of OpenFlow specifications [**?**]. Supporting only a single version simplifies the code as there is no need to accommodate different structures for messages and functionalities that behave differently across

versions of OpenFlow. If we decide to move towards OpenFlow 1.4 and beyond, BOFUSS will continue to support only a single version.

Connection features. BOFUSS supports connections features that allow: (i) use multiple controllers under different roles; (ii) filter messages per connection; and (iii) auxiliary connections in a single controller. The OpenFlow specification is restrictive in the implementation of the first two items, whereas it contains only guidelines for the implementation of the third. In our implementation, the switch supports one additional TCP channel that carries only packet-in messages. Packets arriving from a channel from the controller receive a tag with the type of the channel. This identification is necessary to return a possible reply message through the same channel. For example, the reply from a status request arriving from the main channel returns via the same channel. This initial implementation of auxiliary channels in BOFUSS provides a base for researchers to extend the switch to support extra channels and handle different messages. Any extension would only require adding extra connection listeners, plus defining and handling what kind of OpenFlow message the connection accepts.

2.2.2. Architecture of BOFUSS

We now discuss the structure and organization of BOFUSS, depicted in Figure 1, and how it implements the OpenFlow pipeline. At the same time, we also draw a comparison with the architecture of OVS, illustrated in Figure 2. The aim is to aid readers familiar with the most famous virtual switch to understand the differences to BOFUSS. This introduction is a starting point for adventuring researchers and developers interested in using BOFUSS to develop and test new features. Appendix A points to detailed guides that demonstrate how to add or extend switch functionalities.

Execution space. A clear distinction between the architecture of BOFUSS and OVS is that, in OVS, cached packet processing happens in the kernel. When a packet arrives, OVS parses and looks for cached entries in a single flow

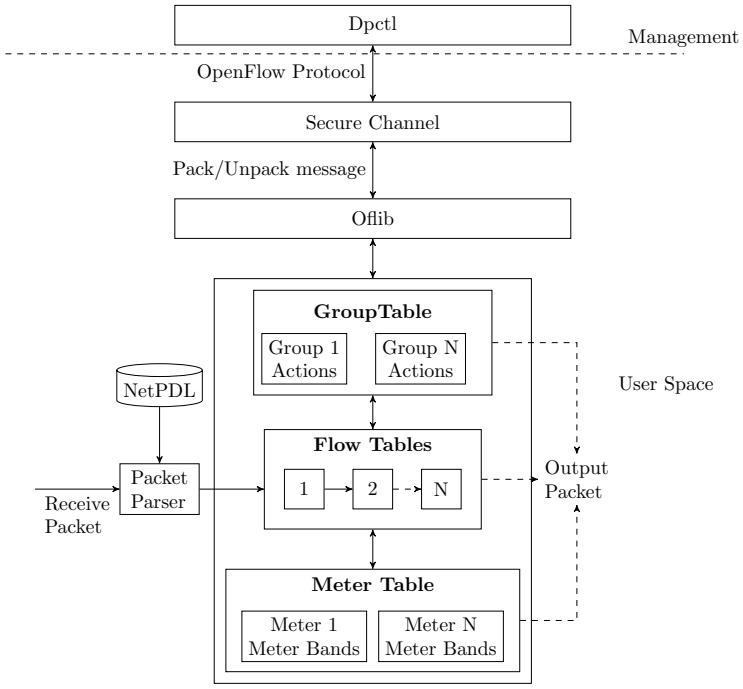

Figure 1: Overview of the architecture of BOFUSS

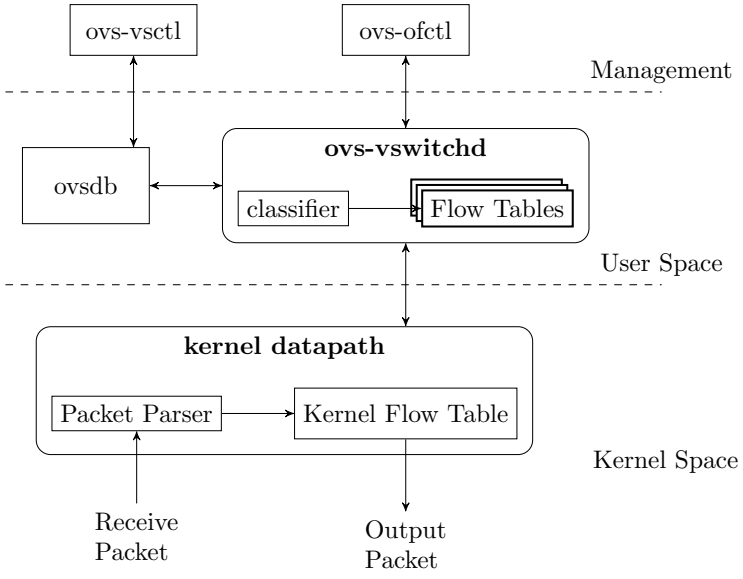

Figure 2: Overview of the architecture of OVS

table. If there is not a recently cached flow entry for the packet, OVS sends the packet to the user-space daemon `ovs-vswitchd`, where it goes through the traditional pipeline of OpenFlow. If a flow matches the packet in user-space, OVS caches the flow in the kernel flow table. In BOFUSS, the whole process happens in user-space, and there is no caching of recently matched flows. To process packets directly on userspace, BOFUSS leverages Linux's packet sockets interface /footnotehttp://man7.org/linux/man-pages/man7/packet.7.html. The sockets allow the switch to receive and send entire packets from any protocol, without changes to its headers.

Packet Parser. A Pipeline packet that comes from the switch ports has the header fields extracted by the Packet Parser first. As previously mentioned, the parsing is automated by the Netbee library [13]. Netbee uses a NetPDL [12] database in the format of eXtensible Markup Language (XML) that contains the definition of the packet headers supported by OpenFlow 1.3. The NetPDL approach has been a powerful component that eases the addition of new fields to OpenFlow, specially in the case of variable headers such as the IPv6 Extension headers [14].

Flow Tables. The search for matching flows in BOFUSS follows a direct approach. The switch's Flow Tables store flows in a linked list, sorted by the priority. Considering the number of flows installed in a table as N and the discussed number to compare two flow's hash maps, the complexity of flow lookup in BOFUSS in $\mathcal{O}(N * M)$. Our simple method is less efficient than OVS's tuple space search classifier [15]. Each Flow Table from OVS may contain multiple hash tables, one for each combination of matching fields in a flow. On lookup, it goes through every hash table, checking for the existence of the hash created from a packet's specific header values. Calling T the number of hash tables in a single Flow Table, the complexity of matching in OVS is $\mathcal{O}(T)$. Since a packet can match different hash tables and flows might have different priorities, the OVS's classifier must search all tables.

Group Table. The Group Table enables different ways to forward packets. It can be used for fast-failover of ports, broadcast and multicast and even to implement Link Aggregation (LAG). The software switch supports all the group entry types defined by the OpenFlow 1.3 specification. We store Group Table entries in a hash map for $\mathcal{O}(1)$ retrieval. The most substantial difference between BOFUSS and OVS groups is the implementation of the type `Select`. Our implementation selects action buckets in a Round-Robin fashion. In this approach, on the first execution of a group, the first bucket is selected. In the next execution, the group chooses the second bucket. The selection moves to the next bucket until the last when it returns to the first. In OVS, the selection uses a hash of the source and destination Ethernet address, VLAN ID, Ethernet type, IPv4/v6 source and destination address, and protocol, plus source and destination ports for TCP and SCTP.

Meter Table. Metering gives the possibility to perform Quality of Service (QoS) on a per-flow basis. The software switch supports the two types available on OpenFlow 1.3, the simple Drop and the Differentiated Services Code Point (DSCP) remark. Both BOFUSS and OVS use the Token Bucket algorithm to measure the per-flow rate and decide if the Meter instruction should be applied or not.

Oflib. This independent library converts OpenFlow messages in a network format to an internal format used by BOFUSS and vice-versa. The process of converting messages is known as pack and unpack. Packing/unpacking a message usually means to add/remove padding bits, but it can also involve the conversion of complex Type-Length-Value (TLV) fields into the most appropriate data structure. One example is the case of the flow match fields, which are translated into hash maps for dynamic and fast access. The Oflib should be the starting point to anyone willing to extend the OpenFlow protocol with new messages.

Secure Channel. The secure channel is a standalone program to set up a con-

nection between the switch and a controller. The division from the datapath happens because OpenFlow does not define the connection method, so implementations are free to define the connection protocol; e.g: Transmission Control Protocol (TCP) or Secure Sockets Layer (SSL); to establish connections. Although having *secure* on its name, at the moment, the component supports only TCP connections. Support for secure oriented protocols, such as SSL, require updates to the Secure Channel code.

Management. The switch includes a command-line tool to perform simple monitoring and management tasks. With `Dpctl`, one can modify and check the current state of switches. A few examples of possible tasks: add new flows, retrieve current flow statistics, and query the state of ports. `ovs-ofctl` plays the same role as `Dpctl` for OVS, a tool for OpenFlow related management. OVS also has an additional tool (`ovs-vsctl`) to manage the switch as a bridge. The use of a database to maintain configuration also enables OVS to restore state if the system goes down or after a software crash. In BOFUSS, we do not support the same capability to restore previous configurations as the switch is only for experimentation purposes.

3. Selected Use Cases

This section presents a series of BOFUSS use cases in which some of the authors have contributed. The nature of these use cases is diverse and can be classified in four types: (1) extensions of the BOFUSS switch, (2) implementation of research ideas, (3) deployment of proof of concepts, and (4) research analysis or teaching SDN architectural concepts. Altogether, they showcase BOFUSS value in supporting industry, research, and academic institutions.

3.1. BEBA

3.1.1. OpenState Extension

BEhavioural BAsed forwarding (BEBA) [16] is a European H2020 project on SDN data plane programmability. The BEBA software prototype has been built

on top of BOFUSS with two main contributions: support for stateful packet forwarding, based on OpenState [17], and packet generation, based on InSPired (InSP) switches [18]. The reason to choose BOFUSS, instead of OVS, was its ease for code modification and portability. Additionally, it was an opportunity to showcase alternative software switches to OVS, and to demonstrate that performance could be greatly enhanced simply by including additional designers and developers to the team.

OpenState is an OpenFlow extension that allows implementing stateful applications in the data plane: the controller configures the switches to autonomously (i.e., without relying on the controller) and dynamically adapt the forwarding behavior. The provided abstraction is based on Finite State Machines where each state defines a forwarding policy and state transitions are triggered by packet-level and time-based events. BOFUSS has been extended using the OpenFlow experimenter framework and adding to each flow table an optional state table to keep track of flow states. Stateful forwarding is enabled thanks to the ability to match on flow state in the flow table and the availability of a data plane action to update the state directly in the fast path. Stateful processing is configured by the controller via experimenter OpenFlow messages.

InSP is an API to define in-switch packet generation operations, which include the specification of triggering conditions, packet format and related forwarding actions. An application example shows how the implementation of an in-switch ARP responder can be beneficial to both the switch and controller scalability.

The additional flexibility introduced by BEBA switches enables several use cases which get benefits from the reduced controller-switch signaling overhead regarding latency and processing. Cascone et. al [19] present an example application showing how BEBA allows implementing a programmable data plane mechanism for network resiliency which provides guaranteed failure detection and recovery delay regardless of controller availability. StateSec [20] is another example of stateful application combining the efficient local monitoring capabilities of BEBA switches with entropy-based algorithm running on the controller

for DDoS Protection.

315 *3.1.2. Performance enhancements*

The second goal for BEBA has been the performance improvement of the data plane. To tackle such a problem, a major refactoring has been put on the field. The set of patches applied to the code base of BOFUSS comprises a Linux kernel bypass to improve the IO network performance, a new design for the 320 packet handle data–type and the full exploitation of the multi-core architectures.

First, the native PF_PACKET Linux socket originally utilized to send/receive packets has been replaced with libpcap [21]. The aim of this refactoring is twofold: on the one hand, it makes the code more portable, on the other, it facilitates the integration with accelerated kernel-bypass already equipped with 325 custom pcap libraries. In order to improve the overall performance, the polling loop adopted by BOFUSS has been replaced by an active polling mechanism. Thanks to the avoidance of the user-space/kernel-space context switching due to the system call `poll()`, the main advantage of this approach is a reduced latency. The downside is a high CPU usage because the process is committed to 330 continuously poll the resource. In real-time system dedicated to network processing, since their main purpose is to maximize performance and not to reduce unnecessary CPU consumption, the choice of an active polling system, such as the one we implemented in the BEBA switch, was significant for obtaining the best results.

335 Second, the structure of the packet-handle has been flattened into a single buffer to replace the multi-chunk design abused in the original code. This change permits to save a dozen of dynamic memory allocations (and related deallocations) on a per-forwarding basis, which represents a remarkable performance improvement per-se.

340 Finally, to tackle the parallelism of the multicore architecture, the PFQ [22] framework was adopted (though it is not active by default). The reason for such a choice over more widely used solution like DPDK is the fine-grained control of the packet–distribution offered by PFQ off-the-shelf. The ability

to dispatch packets to multiple forwarding processes, transparently and with dynamic degrees of flow-consistency, is fundamental to a stateful system like BEBA, where hard consistency guarantees are required by the XFSM programs loaded on the switch.

The remarkable acceleration obtained (nearly 100x) allows the prototype to full switch 4/5 Mpps per–core and to forward the 10G line rate of 64 bytes-long packets with four cores on our 3 GHz but old Xeon architecture. On the other hand, the flexibility of BOFUSS has been in general preserved (in terms for example of modifications to the match-action pipeline and/or addition of new ctrl-switch OpenFlow messages).

A comprehensive description of the various techniques utilized in the BEBA switch, as well as the acceleration contribution of every single patch, are presented in [23].

3.2. AOSS: OpenFlow hybrid switch

AOSS [24] emerged as a solution for the potential scalability problems of using SDN alone to control switch behavior. Its principle is to delegate part of the network intelligence *back* to the network device –or switch–, thus resulting in a hybrid switch. Its implementation is based on the –currently– most common Southbound Interface (SBI) protocol: OpenFlow. The reason to choose BOFUSS was that its code was much more straightforward (and thus faster) to modify and conceptually prove the idea of AOSS.

AOSS accepts proactive installation of OpenFlow rules in the switch and, at the same time, it is capable of forwarding packets through a shortest path when no rule is installed. To create shortest paths, it follows the locking algorithm of All-Path's switches [25], which permits switches to create minimum latency paths on demand, avoiding loops without changing the standard Ethernet frame.

An example of application for AOSS could be a network device that needs to drop some type of traffic (firewall), but forward the rest. In this case, the firewall rules would be installed proactively by the SDN controller and new packets arriving with no associated match would follow the minimum latency

path to destination. This reduces drastically the control traffic, as the SDN controller just needs to bother about the proactive behavior and is not required to reply to `PACKET_IN` messages, usually generated for any unmatched packet.

AOSS is particularly favorable for scenarios as the one described above, but its implementation still does not support composition of applications or reactive SDN behavior. Nevertheless, it is a good approach for hybrid environments where the network intelligence is not strictly centralized, thus improving overall performance.

3.2.1. AOSS Implementation:

To create a PoC of AOSS, different open-source SDN software switches were analyzed. Although OVS was first in the list, due to its kernel-based (and thus higher performance) nature, leveraging its code to quickly build a PoC was laborious. Therefore, the code of BOFUSS was adopted instead. AOSS needs some modifications to generate the hybrid system. The main one requires inserting an autonomous path selection for all packets with no associated match in the OpenFlow table. Fig. 3 reflects these functional changes.

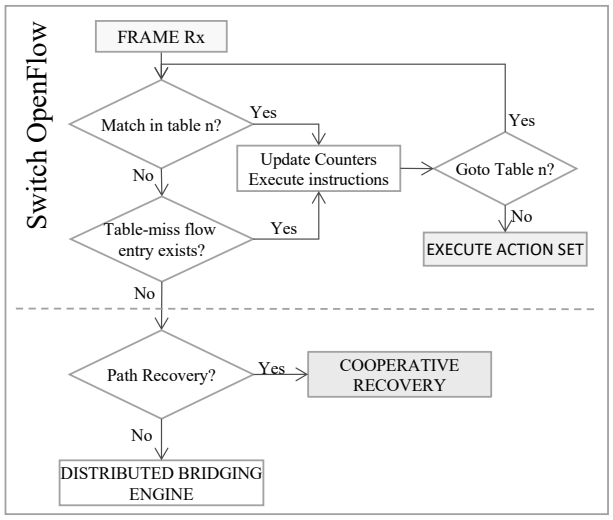

Figure 3: AOSS's Frame Processing [24]

Regarding AOSS implementation, two functional changes and two new functions are defined, as defined in Fig. 4. The first change is a modification in the *Pipeline Process Packet Function* to guarantee compatibility with the autonomous path selection protocol. The second change modifies the drop packet function to create the minimum latency path. As for the new functions, the first is responsible for cleaning the new forwarding tables and the second sends special control frames to allow path recovery after a network failure.

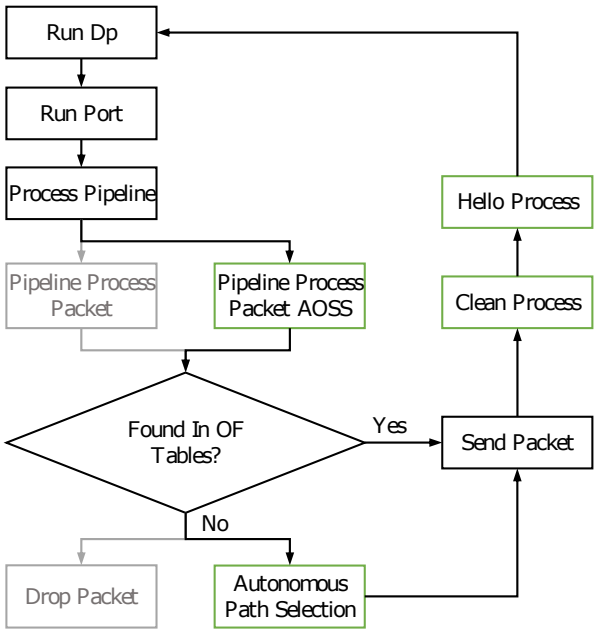

Figure 4: AOSS's Functional Flow Chart

3.3. OnLife: Deploying the CORD project in a national operator

OnLife [26] is a deployment of the CORD project [27] in Telefonica's[1] central offices. The main purpose of OnLife is to bring services as closer to the final user as possible, to enhance their quality, and its first principle is to create a

[1]Main Spanish telecommunications provider

clean network deployment from scratch, with no legacy protocols (e.g. allowing only IPv6 and avoiding IPv4).

The first step in OnLife was building a PoC, purely software-based, to prove its foundations. In CORD, some of the applications in the SDN framework (namely ONOS [28]) require IEEE 802.1ad QinQ tunneling [29] to classify different flows of traffic inside the data center. Therefore BOFUSS was leveraged as OVS does not support this feature.

BOFUSS allowed the initial design of the project, although some initial incompatibilities were found in the communication between ONOS and the switches, solved afterwards. The main conclusion is that BOFUSS became a crucial piece for these deployments, and specific efforts should be made to increase its visibility and community support.

3.4. BOFUSS as a teaching resource

One of the first degrees that teaches the SDN and NFV technologies as tools for the emerging communication networks, specifically 5G networks, is the Master in NFV and SDN for 5G Networks of the University Carlos III of Madrid [30].

BOFUSS is part of the syllabus, presented together with OVS, as one of the two main open source software SDN switches. As its main feature, its easy customization is highlighted. BOFUSS helps explaining the concept of data plane programmability (as a generalization of SDN) and how this is a required feature for future SDN/NFV deployments.

More specifically, the use of BOFUSS is very convenient for introductory labs about networking programmability, as its code is easy to follow and to modify, which can serve as an initial approach for students, to later work with OVS or directly with the P4 language [31]. In fact, it could be leveraged for any networking scenario and not necessarily for SDN/NFV, even to implement and test classic routing protocols such as OSPF. To the best of our knowledge, there is no similar open-source alternative software to achieve these objectives.

Table 1: Classification of works that leverage BOFUSS (part 1/3)

Properties / Article	Description	Type	Why?
OpenState [17]	OpenFlow extension for stateful applications	Research implementation	Pipeline modification
InSP [18]	API to define in-switch packet generation operations	Research implementation	Pipeline modification
AOSS [24]	Stateful (hybrid) SDN switch	Research implementation	Pipeline modification
OPP [32]	Platform-independent stateful in-network processing	Research implementation	Pipeline modification
BPFabric [33, 34]	On-the-fly data plane packet processing pipeline and direct manipulation of network state	Research implementation	Pipeline modification
Fast switchover/failover [35]	New switchover method based on active/active mode (select group)	Research implementation	Pipeline modification
FlowConvertor [36]	Algorithm that provides portability across switch models	Research implementation	Pipeline modification
Chronus [37]	Scheduled consistent network updates	Research implementation	Pipeline modification
REV [38]	New security primitive for SDN	Research implementation	Pipeline modification
RouteFlow [39]	OpenFlow 1.x Dataplane for virtual routing services	Research implementation	OpenFlow version interoperability and Group Tables
TCP connection handover [40]	New method of TCP connection handover in SDN	Research implementation	Modification of OpenFlow 1.3

N/A means *not applicable*.

N/D means *not defined*.

4. Fostering Research & Standardization

Following the classification provided in the previous use cases, this section is devoted to create a brief catalog of the different works found in the literature that have leveraged BOFUSS. The categories are: research implementations or evaluations, PoC implementations, and SDN switch comparatives, and teaching resources. The resulting grouping is summarized in Tables 1, 2 and 3.

4.1. Research implementations or evaluations

Three research implementations have already been introduced in the use cases, namely **OpenState** [17], **InSP** [18] and **AOSS** [24]. All of them envision alternative architectures for SDN in which network switches recover part of the

Table 2: Classification of works that leverage BOFUSS (part 2/3)

Article \ Properties	Description	Type	Why?
Facilitating ICN with SDN [41]	Leveraging SDN for ICN scenarios	Research implementation	Extension of OpenFlow
ÆtherFlow [42]	Application of SDN principles to wireless networks	Research implementation	Extension of OpenFlow
CrossFlow [43],[44]	Application of SDN principles to wireless networks	Research implementation	Extension of OpenFlow
Media Independent Management [45]	Dynamic link information acquisition to optimize networks	Research implementation	Extension of OpenFlow
Automatic failure recovery [46]	Proxy between SDN controller and switches to handle failures	Research implementation	Reuses oflib from ofsoftswitch13
OFSwitch13 [47]	Module to enhance the ns-3 simulator with OpenFlow 1.3	Research implementation	Reuses ofsoftswitch13
Time4 [48]	Approach for network updates (adopted in OpenFlow 1.5)	Research implementation	`Bundle` feature
OFLoad [49]	OF-Based Dynamic Load Balancing for data center networks	Research implementation	OpenFlow `group` option
Blind Packet Forwarding in hierarchical architecture [50]	Implementation of the extended BPF	Research implementation	N/D
GPON SDN Switch [51]	GPON based OpenFlow-enabled SDN virtual switch	Research implementation	Part of the architecture
Traffic classification with stateful SDN [52]	Traffic classification in the data plane to offload the control plane	Research implementation	Leverages OpenState [17] and OPP [32]
Traffic classification and control with stateful SDN [53]	Traffic classification in the data plane to offload the control plane	Research implementation	Leverages OpenState [17]
SPIDER [19]	OpenFlow-like pipeline design for failure detection and fast reroute of traffic flows	Research implementation	Leverages OpenState [17]
StateSec [20]	In-switch processing capabilities to detect and mitigate DDoS attacks	Research implementation	Leverages OpenState [17]
Load balancers evaluation [54]	Evaluation of different load balancer apps	Research evaluation	Leverages OpenState [17]
Recovery of multiple failures in SDN [55]	Comparison of OpenState and OpenFlow in multiple-failure scenarios	Research evaluation	Leverages OpenState [17]

N/A means *not applicable*.

N/D means *not defined*.

Table 3: Classification of works that leverage BOFUSS (part 3/3)

Article \ Properties	Description	Type	Why?
UnifyCore [56]	Mobile architecture implementation in which ofsoftswitch13 is leveraged as a fordwarder	PoC implementation	MAC tunneling
ADN [57]	Architecture that provides QoS on an application flow basis	PoC implementation	Full support of OpenFlow 1.3 (meters and groups `all`/`select`)
TCP connection handover for hybrid honeypot systems [58]	TCP connection handover mechanism implemented in SDN	PoC implementation	Data plane programmability
Multiple Auxiliary TCP/UDP Connections in SDN [59]	Analysis and implementation of multiple connections in SDN	PoC implementation	Extension of OFSwitch13 [47]
State-based security protection mechanisms in SDN [60]	Demonstration of the SDN Configuration (CFG) protection	PoC implementation	Leverages OpenState [17]
Advanced network functions [61]	Stateful data-plane network functions	PoC implementation	Leverages OPP [32]
PathMon [62]	Granular traffic monitoring	PoC implementation	N/D
QoT Estimator in SDN-Controlled ROADM networks [63]	Implementation of a QoT estimator in a simulated optical network	PoC implementation	N/D
OPEN PON [64]	Integration of 5G core and optical access networks	PoC implementation (MSc Thesis)	Support of IEEE 1904.1 SIEPON, meters and Q-in-Q
Stochastic Switching Using OpenFlow [65]	Analysis and implementation of stochastic routing in SDN	PoC implementation (MSc Thesis)	`Select` function of `Group` feature
OpenFlow forwarders [66]	Routing granularity in OpenFlow 1.0 and 1.3	SDN switch comparative	N/A
Open source SDN [67]	Performance of open source SDN virtual switches	SDN switch comparative	N/A
Visual system to learn OF [68]	A visual system to support learning of OpenFlow-based networks	Teaching resource	N/D

N/A means *not applicable*.

N/D means *not defined*.

intelligence of the network and, accordingly, they leverage BOFUSS thanks to its easily modifiable pipeline.

Also based on pipeline modifications, **Open Packet Processor (OPP)** [32] enhances the approach of OpenState to support extended Finite State Machines, which broadens the potential functionality of the data plane. **BPFabric** [33, 34] defines an architecture that allows instantiating and querying, on-the-fly, the packet processing pipeline in the data plane.

Regarding the evolution of current SBI protocols (namely OpenFlow), an alternative switchover procedure (active/active instead of active/standby) is presented in [35], which leverages the `select` group of BOFUSS. **RouteFlow** [39] is a pioneering architectural proposal to deliver flexible (virtual) IP routing services over OpenFlow networks [69] (developed by the same core research group at CPqD behind BOFUSS), which extensively used the software switch for fast prototyping, interoperability tests with OpenFlow 1.2 and 1.3, and new features such as group tables.

Considering the heterogeneity of switch pipeline implementations, **FlowConvertor** [36] defines an algorithm that provides portability across different models. To prove the idea, it applies it to a BOFUSS switch, as it demonstrates to have a flexible and programmable pipeline. Another research topic in relation to the SBI are transactional operations and consistent network updates (currently OpenFlow does not support these types of procedures), and **Chronus** [37] modifies BOFUSS to provide scheduled network updates, to avoid potential problems, such as communication loops or blackholes. Finally, **REV** [38] designs a new security primitive for SDN, specifically aimed to prevent rule modification attacks.

In the specific case of enhancements of OpenFlow, an extension of OpenFlow 1.3 thanks to BOFUSS is introduced in [40], which includes two new actions (`SET_TCP_ACK` and `SET_TCP_SEQ`) to modify the ACK and SEQ values in TCP connections. Alternatively, the matching capabilities of OpenFlow have been extended in [41] to provide an optimal parsing of packets in the context of Information-Centric Networking (ICN). Both **ÆtherFlow** [42] and **Cross-**

Flow [43] study how to evolve OpenFlow to include the SDN principles in wireless networks. In this regard, BOFUSS acts as an OpenFlow agent with custom extensions. Another extension of OpenFlow is provided in [45], were the authors design a framework where the key is media independent management.

Different research implementations are based on BOFUSS because they simply wanted to leverage some piece of its code. For example, the automatic failure mechanism described in [46] reuses the `oflib` library. **OFSwitch13** [47] reuses the whole code of BOFUSS to incorporate the support of OpenFlow 1.3 in the network simulator ns-3. **Time4** [48] reuses the `bundle` feature to implement an approach for network updates (actually adopted in OpenFlow 1.5). **OFLoad** [49] leverages the OpenFlow `group` option from BOFUSS to design an strategy for dynamic load balancing in SDN. The principles of Blind Packet Forwarding (BPF) also reuse the code of BOFUSS for the implementation. A textbfGPON SDN Switch, where BOFUSS is part of the architecture, is also designed and developed in [51].

Finally, several research ideas leverage OpenState and, thus, BOFUSS. The first two were already mentioned previously: **SPIDER** [19] and **StateSec** [20], both examples of stateful applications aimed to provide enhanced network resiliency and monitoring, respectively. Also, **traffic classificators** based on OpenState are also presented in [52] and [53]. Additionally, an evaluation of SDN load balancing implementations is performed in [54], and authors in [55] compare recovery of SDN from multiple failures for OpenFlow vs. OpenState.

4.2. PoC implementations

BOFUSS has also been part of different PoC implementations. For example, **UnifyCore** [56] is an integrated mobile network architecture, based on OpenFlow but leveraging legacy infrastructure. They evaluate the MAC tunneling implemented in BOFUSS with `iperf`. **ADN** [57] describes an architecture that provides QoS based on application flow information, and they chose BOFUSS because it fully supports OpenFlow 1.3. Authors in [58] implemented a novel TCP connection handover mechanism with BOFUSS, aimed to provide trans-

parency to honeypots by generating the appropriate sequence and acknowledgement numbers for the TCP redirection mechanism to work.

One PoC leveraged OFSwitch13 (BOFUSS in ns-3) to support multiple transport connections in SDN simulations [59], while authors in [60] leverage OpenState to demonstrate that stateful data-plane designs can provide additional security for operations such as link reconfiguration or switch identification. Advanced network functions based on OPP are implemented and tested in [61].

Out of curiosity, there are some works that use BOFUSS just as the SDN software switch for no particular reason (as many others use OVS by default). One of them is **PathMon** [62], which provides granular traffic monitoring. Another one is a QoT estimator for ROADM networks implemented and evaluated in [63].

Finally, two MSc. Thesis have also be developed based on BOFUSS. The first one is **OPEN PON** [64], which analyzes the integration between the 5G core and optical access networks. BOFUSS was selected because of different reasons, but mainly because of its support of standards, such as Q-in-Q (required to emulate the behaviour of the OLT modules), which is not properly implemented in OVS. The second one describes stochastic switching using OpenFlow [65] and BOFUSS was once again chosen due to its good support of specific features, such as the `select` function.

4.3. Comparative reports and Teaching resources

In this last category, it is worth mentioning two comparison studies: a performance analysis of OpenFlow forwarders based on routing granularity [66], and an experimental analysis of different pieces of software in an SDN open source environment [67]. The former compares BOFUSS with other switches, while the latter analyzes the role of BOFUSS in a practical SDN framework. Finally, a nice teaching resource is described in [68], where the authors present a system they put in practice to learn the basics of OpenFlow in a visual manner.

5. Evaluation

As previously stated, there are currently two main types of software switches for SDN environments: OVS and BOFUSS. The main conclusion is that OVS performs much better, but it is hard to modify, while BOFUSS is particularly suitable for customizations and research work, even though its throughput limitations. This is just a qualitative comparison.

For this reason, in this section, we provide an additional quantitative evaluation for OVS vs. BOFUSS. More specifically, we will compare OVS with the two main *flavours* of BOFUSS, namely the **original** BOFUSS [7] and the **enhanced** version implemented by the *BEBA* [70] project. The comparison will be performed via two tests:

1. Individual benchmarking of the three switches via iPerf [71]
2. Evaluation in a data center scenario with characterized traffic and three different networks comprised of the different types of switches

The main purpose is to provide a glance at the performance of BOFUSS, which might be good enough for many research scenarios, even if OVS exhibits better results overall[2].

5.1. Individual benchmarking

For this first test, we directly benchmarked each of the three switches (OVS and the two flavours of BOFUSS) with iPerf [71]. Our hardware infrastructure consisted of 1 computer powered by Intel(R) Core(TM) i7 processors (3,4 GHz), 8 CPU cores, with 24 GB of RAM and Ubuntu 14.04 as Operating System. We deployed one single switch of each type and run iPerf 10 times for each scenario, obtaining two parameters: throughput and packet processing delay, both represented with their average value and standard deviation. Additionally, we measured the CPU usage of each switch with flows of different throughput

[2] A comparison of OVS with other software switches, but without including BOFUSS, is provided in [72].

Table 4: Throughput and packet processing delay of the three individual types of software switches, measured with iPerf

Switch	Throughput	Packet Processing Delay
OVS	51.413 Gbps ±2.6784	724.16 ns ±32.23
Enhanced BOFUSS	1.184 Gbps $\pm 3.945*10^{-3}$	2354.48 ns ±1.4599
Original BOFUSS	0.186 Gbps $\pm 6.86*10^{-5}$	59115.75 ns ±130.88

(1, 10 , 50, 100, 500, 1000 Mbps, and no restriction (MAX)). The value of CPU usage has been obtained with the `top` Linux command.

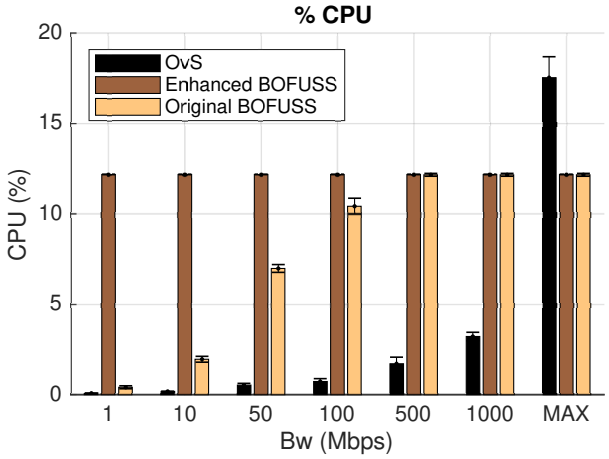

Figure 5: Percentage of CPU usage for each switch type

The results for the throughput and packet processing delay are shown in Table 4. Although OVS outperforms BOFUSS, it is important to notice how the enhanced switch has a tolerable packet processing delay and it surpasses 1 Gbps, a result considered a reasonable throughput for most common networking scenarios.

As for the CPU usage, it is represented by Fig. 5. Both BOFUSS and its enhanced version are, by default, monolithic (they just leverage one core) and consume up to a 12.5% of CPU, which is exactly an eighth of the total available CPU, while OVS uses more than one core and, hence, its consumption might be higher. Finally, we would like to highlight two aspects: (1) enhanced BOFUSS

always consumes the maximum CPU, which is stated and explained in Section 3.1.2, and (2) the MAX value for every switch differs and it is directly related with the values from Table 4.

5.2. Evaluation in a data center scenario

For this second test, we focused on realistic scenarios data center deployments, where software switches could an essential part of the network infrastructure. We built a *Spine-Leaf* topology [73, 74, 75], typically deployed for data center networks. More specifically, a 4-4-20 Spine-Leaf with 2 rows of 4 switches (4 of type *spine* and 4 of type *leaf*) and 20 servers per leaf switch for a total of 80 servers, as illustrated in Fig. 6.

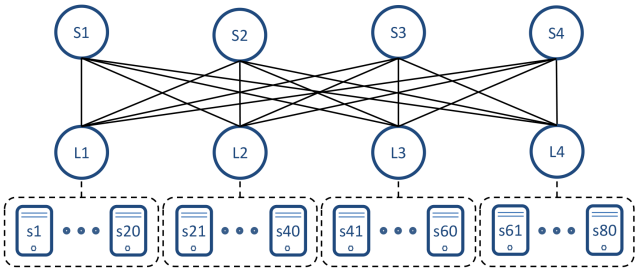

Figure 6: Spine-Leaf 4-4-20 evaluation topology [76]

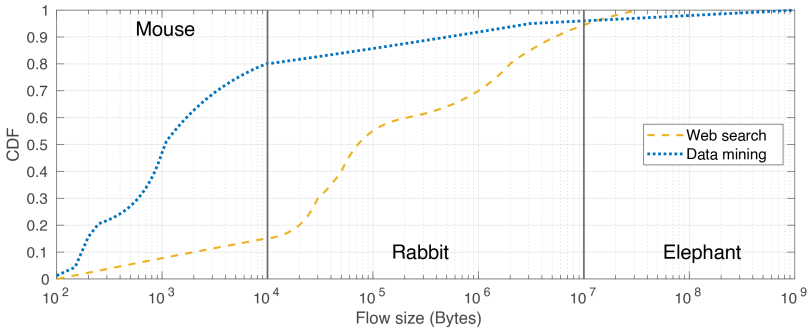

Figure 7: Flow size distributions [76]

To emulate data center-like traffic, we developed a customized traffic generator [76]. This generator implements two different flow size distributions, namely

Table 5: Experimental setup of the data center scenarios

Parameter	Value
Network topology	Spine-Leaf (4 - 4)[73]
Servers per leaf switch	20
Flow distribution	Random inter-leaf
Flow size distributions	Web search[77] & Data mining [78]
Network offered load (%)	10, 20 & 40%
Link speed (Mpbs)	100Mbps
Run length (s)	1800 s
Warm up time (s)	800 s
Number of runs	10

Data Mining and *Web Search*, derived from experimental traces taken from actual data center networks [77, 78]. Figure 7 shows the cumulative distribution function (CDF) of both distributions and also illustrates how flows are classified according to their size. Flows with less than 10 KB and more than 10 MB of data are considered *mouse* and *elephant* flows, respectively, as explained in [78]. The remaining flows are identified as *rabbit* flows. Traffic flows are randomly distributed between any pair of servers attached to two different leaf switches with no further restrictions.

Our hardware infrastructure consisted of a cluster of 5 computers powered by Intel(R) Core(TM) i7 processors (4,0 GHz) with 24 GB of RAM and Ubuntu 14.04 as Operating System, all of which are interconnected via a GbE Netgear GS116 switch. Each experiment was executed for 1800 seconds and repeated 10 times to compute 95% confidence intervals. Additionally, we considered a warm-up time of 800 seconds to mitigate any transitory effect on the results. Table 5 summarizes the full setup of the conducted experiments.

To evaluate the performance of OVS and the two flavours of BOFUSS, we measured throughput and flow completion time, which are depicted in Fig. 8

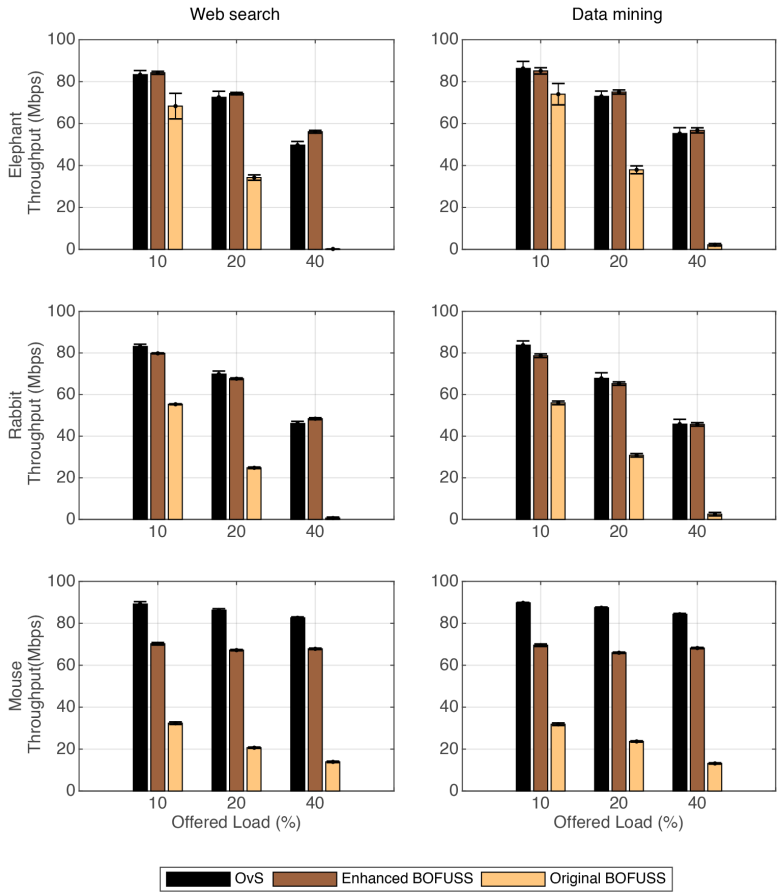

Figure 8: Throughput in the Spine-Leaf topology for each switch type

and Fig. 9, respectively[3]. The graphs are divided into the three types of flows, and we evaluated an increasing network offered load of 10%, 20% and 40%. The results show that OVS and the enhanced BOFUSS perform quite similarly. In fact, they provide almost the same results for the elephants and rabbit flows (even more favorable for the enhanced BOFUSS in some cases), and better for OVS in the case of the mouse flows. In all cases, the original BOFUSS is outperformed by OVS and the enhanced BOFUSS. In fact, when the offered

[3]Raw evaluation data can be found at [79].

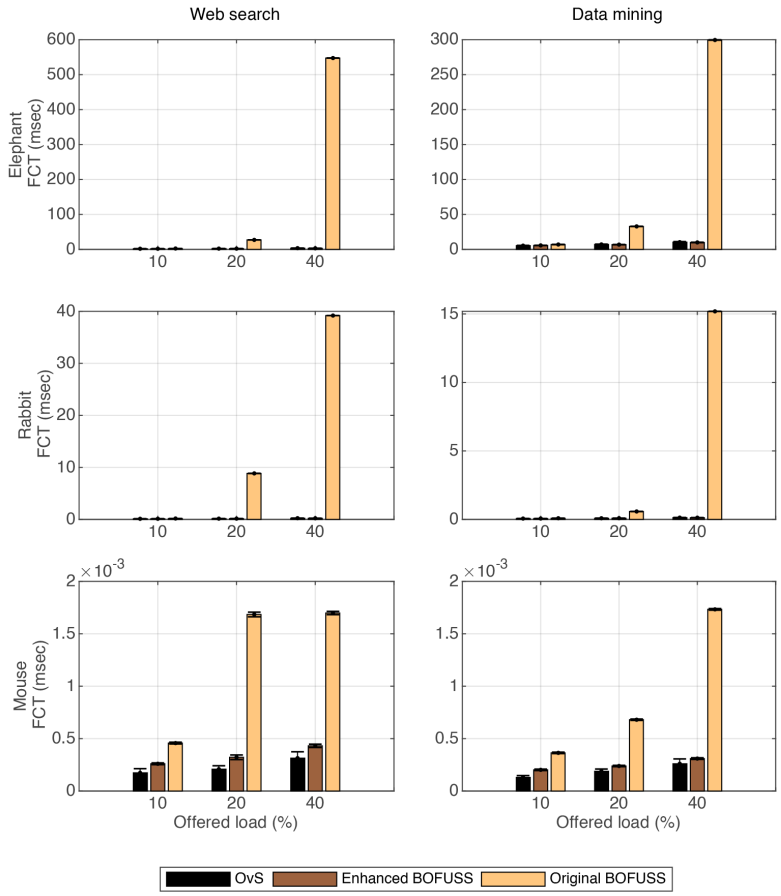

Figure 9: Flow Completion Time in the Spine-Leaf topology for each switch type

load reaches the 40%, the results are particularly bad for original BOFUSS, which is mainly overload by the biggest flows (elephant and rabbit), obtaining almost a null throughput. Finally, it is important to highlight that the enhanced BOFUSS shows smaller standard deviations than OVS, although the values of OVS are not bad either.

The main conclusion of this second test is the enhancements provided by BEBA make BOFUSS a feasible option for experiments dependent on higher performance. Indeed, the results of the BOFUSS switch are comparable to OVS, reinforcing it as a reasonable option when modifications in the switch are

required, or even when some features of OpenFlow are needed and not available in OVS.

6. Conclusions and Future Work

During the article, we have provided a guided overview of BOFUSS, trying to portray the importance of this software switch in SDN environments, which are pivotal towards next-generation communication networks. We first introduced the history of the switch and presented its architectural design. Secondly, we described a set of selected use cases that leverage BOFUSS for diverse reasons: from easy customization to features missing in OVS. The purpose was to highlight that, although OVS may be thought as the king of software switches, BOFUSS can also be a good candidate for specific scenarios where OVS is too complex (or almost impossible) to play with. Afterwards, we complemented the selected use cases with a comprehensive survey of works that also use BOFUSS, remarkable when the switch did not even had an official name and publication. Finally, we carried out an evaluation of BOFUSS vs. OVS to prove that our switch has also a reasonable performance, greatly improved since the release of the original project. Researchers looking for a customized switch should carefully analyze the tradeoff between complexity and performance in OVS and BOFUSS.

As future lines of work, we envision the growth of the community around BOFUSS and newer contributions for the switch. For this purpose, we have created a set of comprehensive guides, listed in Appendix Appendix A, to solve and help the work for researchers interested in the switch. Regarding the evolution of SBI protocols, the specifications of OpenFlow is currently stuck and the ONF is focusing now on the advanced programmability provided by the P4 language [31] and P4 Runtime. Therefore, BOFUSS could join its efforts towards the adoption of this new protocol. In any case, we welcome any questions, suggestions or ideas to keep the BOFUSS community alive, and to do so, you can directly contact the team at the GitHub repository stated in [7].

Acknowledgment

This work was partially supported by Ericsson Innovation Center in Brazil. Additional support is provided by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) grant numbers 310317/2013-4 and 310930/2016-2, by grants from Comunidad de Madrid through project TAPIR-CM (S2018/TCS-4496), and by the University of Alcala through project CCGP2017-EXP/001 and the "Formación del Profesorado Universitario (FPU)" program.

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE 103 (1) (2015) 14–76. doi:10.1109/JPROC.2014.2371999.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74. doi:10.1145/1355734.1355746.
URL http://doi.acm.org/10.1145/1355734.1355746

[3] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, The Design and Implementation of Open vSwitch, in: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, USENIX Association, Berkeley,CA,USA, 2015, pp. 117–130.
URL http://dl.acm.org/citation.cfm?id=2789770.2789779

[4] E. L. Fernandes, C. E. Rothenberg, OpenFlow 1.3 software switch, Salao de Ferramentas do XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos SBRC (2014) 1021–1028.

[5] OpenFlow 1.1 Software Switch.
URL https://github.com/TrafficLab/of11softswitch

[6] Stanford OpenFlow Reference Switch repository.
URL http://yuba.stanford.edu/git/gitweb.cgi?p=openflow.git;a=summary

[7] OpenFlow 1.3 switch - CPqD/ofsoftswitch13.
URL https://github.com/CPqD/ofsoftswitch13

[8] OpenFlow Python Switch repositorrty.
URL https://github.com/floodlight/oftest

[9] Open vSwitch.
URL http://openvswitch.org/

[10] OpenFlow Software Switch 1.1 announcement.
URL https://mailman.stanford.edu/pipermail/openflow-discuss/2011-May/002183.html

[11] OpenFlow 1.2 Toolkit announcement.
URL https://mailman.stanford.edu/pipermail/openflow-discuss/2012-July/003479.html

[12] F. Risso, M. Baldi, NetPDL: an extensible XML-based language for packet header description, Computer Networks 50 (5) (2006) 688–706.

[13] NetBee.
URL https://github.com/netgroup-polito/netbee

[14] R. R. Denicol, E. L. Fernandes, C. E. Rothenberg, Z. L. Kis, On IPv6 support in OpenFlow via flexible match structures, OFELIA/CHANGE Summer School.

[15] V. Srinivasan, S. Suri, G. Varghese, Packet Classification Using Tuple Space Search, in: L. Chapin, J. P. G. Sterbenz, G. M. Parulkar, J. S. Turner (Eds.), Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 30 - September 3, 1999, Cambridge, Massachusetts, USA,

ACM, 1999, pp. 135–146. doi:10.1145/316188.316216.
URL https://doi.org/10.1145/316188.316216

[16] BEBA Behavioral Based Forwarding.
URL http://http://www.beba-project.eu/

[17] G. Bianchi, M. Bonola, A. Capone, C. Cascone, OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch, SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 44–51. doi:10.1145/2602204.2602211.
URL http://doi.acm.org/10.1145/2602204.2602211

[18] R. Bifulco, J. Boite, M. Bouet, F. Schneider, Improving SDN with InSPired Switches, in: Proceedings of the Symposium on SDN Research, SOSR '16, ACM, New York,NY,USA, 2016, pp. 11:1–11:12. doi:10.1145/2890955.2890962.
URL http://doi.acm.org/10.1145/2890955.2890962

[19] C. Cascone, D. Sanvito, L. Pollini, A. Capone, B. Sans, Fast failure detection and recovery in SDN with stateful data plane, International Journal of Network Management 27 (2) (2017) e1957–n/a, e1957 nem.1957. doi:10.1002/nem.1957.
URL http://dx.doi.org/10.1002/nem.1957

[20] J. Boite, P. A. Nardin, F. Rebecchi, M. Bouet, V. Conan, Statesec: Stateful monitoring for DDoS protection in software defined networks, in: 2017 IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–9. doi:10.1109/NETSOFT.2017.8004113.

[21] TCPDUMP/LIBPCAP public repository.
URL https://www.tcpdump.org/

[22] N. Bonelli, S. Giordano, G. Procissi, Network Traffic Processing With PFQ, IEEE Journal on Selected Areas in Communications 34 (6) (2016) 1819–1833.

[23] N. Bonelli, G. Procissi, D. Sanvito, R. Bifulco, The acceleration of OfSoftSwitch, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 1–6. doi:10.1109/NFV-SDN.2017.8169842.

[24] J. Alvarez-Horcajo, I. Martinez-Yelmo, E. Rojas, J. A. Carral, D. Lopez-Pajares, New cooperative mechanisms for software defined networks based on hybrid switches, Transactions on Emerging Telecommunications Technologies (2017) e3150–n/aE3150 ett.3150. doi:10.1002/ett.3150.
URL http://dx.doi.org/10.1002/ett.3150

[25] E. Rojas, G. Ibanez, J. M. Gimenez-Guzman, J. A. Carral, A. Garcia-Martinez, I. Martinez-Yelmo, J. M. Arco, All-Path bridging: Path exploration protocols for data center and campus networks, Computer Networks 79 (Supplement C) (2015) 120 – 132. doi:https://doi.org/10.1016/j.comnet.2015.01.002.

[26] R. S. Montero, E. Rojas, A. A. Carrillo, I. M. Llorente, Extending the Cloud to the Network Edge, Computer 50 (4) (2017) 91–95. doi:doi.ieeecomputersociety.org/10.1109/MC.2017.118.

[27] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, W. Snow, Central office re-architected as a data center, IEEE Communications Magazine 54 (10) (2016) 96–101. doi:10.1109/MCOM.2016.7588276.

[28] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar, ONOS: Towards an Open,Distributed SDN OS, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York,NY,USA, 2014, pp. 1–6. doi:10.1145/2620728.2620744.
URL http://doi.acm.org/10.1145/2620728.2620744

[29] 802.1ad - Provider Bridges.
URL http://www.ieee802.org/1/pages/802.1ad.html

[30] Master in NFV and SDN for 5G Networks. UC3M.
URL https://www.uc3m.es/master/NFV-SDN-5g-networks

[31] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming Protocol-independent Packet Processors, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95. doi:10.1145/2656877.2656890.

[32] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, C. Cascone, Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing, ArXiv e-printsarXiv:1605.01977.
URL http://adsabs.harvard.edu/abs/2016arXiv160501977B

[33] S. Jouet, R. Cziva, D. P. Pezaros, Arbitrary packet matching in OpenFlow, in: 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR), 2015, pp. 1–6. doi:10.1109/HPSR.2015.7483106.

[34] S. Jouet, D. P. Pezaros, BPFabric: Data Plane Programmability for Software Defined Networks, in: Proceedings of the Symposium on Architectures for Networking and Communications Systems, ANCS '17, IEEE Press, Piscataway,NJ,USA, 2017, pp. 38–48. doi:10.1109/ANCS.2017.14.
URL https://doi.org/10.1109/ANCS.2017.14

[35] K. Nguyen, Q. T. Minh, S. Yamada, Novel fast switchover on OpenFlow switch, in: 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), 2014, pp. 543–544. doi:10.1109/CCNC.2014.6940510.

[36] H. Pan, G. Xie, Z. Li, P. He, L. Mathy, FlowConvertor: Enabling portability of SDN applications, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057135.

[37] J. Zheng, G. Chen, S. Schmid, H. Dai, J. Wu, Q. Ni, Scheduling Congestion- and Loop-Free Network Update in Timed SDNs, IEEE Journal on Selected Areas in Communications 35 (11) (2017) 2542–2552. doi:10.1109/JSAC.2017.2760146.

[38] P. Zhang, Towards rule enforcement verification for software defined networks, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8056994.

[39] A. Vidal12, F. Verdi, E. L. Fernandes, C. E. Rothenberg, M. R. Salvador, Building upon RouteFlow: a SDN development experience.

[40] A. Binder, T. Boros, I. Kotuliak, A SDN Based Method of TCP Connection Handover, Springer International Publishing, Cham, 2015, pp. 13–19. doi:10.1007/978-3-319-24315-3_2.

[41] P. Zuraniewski, N. van Adrichem, D. Ravesteijn, W. IJntema, C. Papadopoulos, C. Fan, Facilitating ICN Deployment with an Extended Openflow Protocol, in: Proceedings of the 4th ACM Conference on Information-Centric Networking, ICN '17, ACM, New York,NY,USA, 2017, pp. 123–133. doi:10.1145/3125719.3125729.
URL http://doi.acm.org/10.1145/3125719.3125729

[42] M. Yan, J. Casey, P. Shome, A. Sprintson, A. Sutton, ÆtherFlow: Principled Wireless Support in SDN, in: 2015 IEEE 23rd International Conference on Network Protocols (ICNP), 2015, pp. 432–437. doi:10.1109/ICNP.2015.9.

[43] P. Shome, M. Yan, S. M. Najafabad, N. Mastronarde, A. Sprintson, CrossFlow: A cross-layer architecture for SDR using SDN principles, in: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015, pp. 37–39. doi:10.1109/NFV-SDN.2015.7387403.

[44] P. Shome, J. Modares, N. Mastronarde, A. Sprintson, Enabling Dynamic Reconfigurability of SDRs Using SDN Principles, in: Ad Hoc Networks, Springer, 2017, pp. 369–381.

[45] C. Guimares, D. Corujo, R. L. Aguiar, Enhancing openflow with Media Independent Management capabilities, in: 2014 IEEE International Conference on Communications (ICC), 2014, pp. 2995–3000. doi:10.1109/ICC.2014.6883780.

[46] M. Kuźniar, P. Perešíni, N. Vasić, M. Canini, D. Kostić, Automatic failure recovery for software-defined networks, in: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, 2013, pp. 159–160.

[47] L. J. Chaves, I. C. Garcia, E. R. M. Madeira, OFSwitch13: Enhancing Ns-3 with OpenFlow 1.3 Support, in: Proceedings of the Workshop on Ns-3, WNS3 '16, ACM, New York,NY,USA, 2016, pp. 33–40. doi:10.1145/2915371.2915381.
URL http://doi.acm.org/10.1145/2915371.2915381

[48] T. Mizrahi, Y. Moses, Time4: Time for SDN, IEEE Transactions on Network and Service Management 13 (3) (2016) 433–446. doi:10.1109/TNSM.2016.2599640.

[49] R. Trestian, K. Katrinis, G. M. Muntean, OFLoad: An OpenFlow-Based Dynamic Load Balancing Strategy for Datacenter Networks, IEEE Transactions on Network and Service Management 14 (4) (2017) 792–803. doi:10.1109/TNSM.2017.2758402.

[50] I. Simsek, Y. I. Jerschow, M. Becke, E. P. Rathgeb, Blind Packet Forwarding in a hierarchical architecture with Locator/Identifier Split, in: 2014 International Conference and Workshop on the Network of the Future (NOF), 2014, pp. 1–5. doi:10.1109/NOF.2014.7119775.

[51] S. S. W. Lee, K. Y. Li, M. S. Wu, Design and Implementation of a GPON-Based Virtual OpenFlow-Enabled SDN Switch, Journal of Lightwave Technology 34 (10) (2016) 2552–2561. doi:10.1109/JLT.2016.2540244.

[52] D. Sanvito, D. Moro, A. Capone, Towards traffic classification offloading to stateful SDN data planes, in: 2017 IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–4. doi:10.1109/NETSOFT.2017.8004227.

[53] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, T. Zhang, On-the-fly traffic classification and control with a stateful SDN approach, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6. doi:10.1109/ICC.2017.7997297.

[54] W. J. A. Silva, K. L. Dias, D. F. H. Sadok, A performance evaluation of Software Defined Networking load balancers implementations, in: 2017 International Conference on Information Networking (ICOIN), 2017, pp. 132–137. doi:10.1109/ICOIN.2017.7899491.

[55] M. S. M. Zahid, B. Isyaku, F. A. Fadzil, Recovery of Software Defined Network from Multiple Failures: Openstate Vs Openflow, in: 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), 2017, pp. 1178–1183. doi:10.1109/AICCSA.2017.32.

[56] M. Nagy, I. Kotuliak, J. Skalny, M. Kalcok, T. Hirjak, Integrating Mobile OpenFlow Based Network Architecture with Legacy Infrastructure, Springer International Publishing, Cham, 2015, pp. 40–49. doi:10.1007/978-3-319-24315-3_5.

[57] F. S. Tegueu, S. Abdellatif, T. Villemur, P. Berthou, T. Plesse, Towards application driven networking, in: 2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2016, pp. 1–6. doi:10.1109/LANMAN.2016.7548865.

[58] W. Fan, D. Fernandez, A novel SDN based stealthy TCP connection handover mechanism for hybrid honeypot systems, in: 2017 IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–9. doi:10.1109/NETSOFT.2017.8004194.

[59] H. Yang, C. Zhang, G. Riley, Support Multiple Auxiliary TCP/UDP Connections in SDN Simulations Based on Ns-3, in: Proceedings of the Workshop on Ns-3, WNS3 '17, ACM, New York,NY,USA, 2017, pp. 24–30. doi:10.1145/3067665.3067670.
URL http://doi.acm.org/10.1145/3067665.3067670

[60] T. Arumugam, S. Scott-Hayward, Demonstrating state-based security protection mechanisms in software defined networks, in: 2017 8th International Conference on the Network of the Future (NOF), 2017, pp. 123–125. doi:10.1109/NOF.2017.8251231.

[61] M. Bonola, R. Bifulco, L. Petrucci, S. Pontarelli, A. Tulumello, G. Bianchi, Demo: Implementing advanced network functions with stateful programmable data planes, in: 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2017, pp. 1–2. doi:10.1109/LANMAN.2017.7972183.

[62] M.-H. Wang, S.-Y. Wu, L.-H. Yen, C.-C. Tseng, PathMon: Path-specific traffic monitoring in OpenFlow-enabled networks, in: 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), 2016, pp. 775–780. doi:10.1109/ICUFN.2016.7537143.

[63] A. A. Díaz-Montiel, J. Yu, W. Mo, Y. Li, D. C. Kilper, M. Ruffini, Performance analysis of QoT estimator in SDN-controlled ROADM networks, in: 2018 International Conference on Optical Network Design and Modeling (ONDM), 2018, pp. 142–147. doi:10.23919/ONDM.2018.8396121.

[64] M. D. M. Silva, N. Filipe, OPEN PON: Seamless integration between 5G core and optical access networks, Master's thesis, Universitat Politècnica de Catalunya (2016).

[65] K. Shahmir Shourmasti, Stochastic Switching Using OpenFlow, Master's thesis, Institutt for telematikk (2013).

[66] V. Šulák, P. Helebrandt, I. Kotuliak, Performance analysis of OpenFlow forwarders based on routing granularity in OpenFlow 1.0 and 1.3, in: 2016 19th Conference of Open Innovations Association (FRUCT), 2016, pp. 236–241. doi:10.23919/FRUCT.2016.7892206.

[67] K. Tantayakul, R. Dhaou, B. Paillassa, W. Panichpattanakul, Experimental analysis in SDN open source environment, in: 2017 14th International Conference on Electrical Engineering/Electronics,Computer,Telecommunications and Information Technology (ECTI-CON), 2017, pp. 334–337. doi:10.1109/ECTICon.2017.8096241.

[68] H. Fujita, Y. Taniguchi, N. Iguchi, A system to support learning of OpenFlow network by visually associating controller configuration information and logical topology, in: 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), 2017, pp. 1–3. doi:10.1109/GCCE.2017.8229319.

[69] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, R. Raszuk, Revisiting Routing Control Platforms with the Eyes and Muscles of Software-defined Networking, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 13–18. doi:10.1145/2342441.2342445.
URL http://doi.acm.org/10.1145/2342441.2342445

[70] BEBA Software Switch.
URL https://github.com/ccascone/beba-switch

[71] iPerf - The TCP, UDP and SCTP network bandwidth measurement tool.
URL https://iperf.fr/

[72] V. Fang, T. Lvai, S. Han, S. Ratnasamy, B. Raghavan, J. Sherry, Evaluating Software Switches: Hard or Hopeless?, Tech. Rep. UCB/EECS-2018-136,

EECS Department, University of California, Berkeley (Oct 2018).
URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-136.html

[73] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, e. a. Yadav, CONGA:Distributed Congestion-aware Load Balancing for Datacenters, SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 503–514.

[74] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, A. Akella, Presto:Edge-based Load Balancing for Fast Datacenter Networks, SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 465–478. doi:10.1145/2829988.2787507.

[75] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, pFabric:Minimal Near-optimal Datacenter Transport, SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 435–446. doi:10.1145/2534169.2486031.

[76] J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, I. Martinez-Yelmo, TCP-path: Improving load balance by network exploration, in: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), 2017, pp. 1–6. doi:10.1109/CloudNet.2017.8071533.

[77] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), SIGCOMM Comput. Commun. Rev. 41 (4) (2010) 63–74.

[78] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta, VL2:a scalable and flexible data center network, SIGCOMM Comput. Commun. Rev. 39 (4) (2009) 51–62.

[79] J. Alvarez-Horcajo, BOFUSS raw evaluation data.
URL https://github.com/gistnetserv-uah/GIST-DataRepo/tree/master/BOFUSS-spine_leaf-190120

Appendix A. Resources for Researchers and Developers

- **Overview of the Switch's Architecture**.
  https://github.com/CPqD/ofsoftswitch13/wiki/Overview-of-the-Switch's-Architecture
- **Implementation Details**.
  https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-Implementation-Details
- **How to Add a New OpenFlow Message**.
  https://github.com/CPqD/ofsoftswitch13/wiki/Adding-New-OpenFlow-Messages
- **How to Add a New Matching Field**
  https://github.com/CPqD/ofsoftswitch13/wiki/Adding-a-New-Match-Field
- **Frequently Asked Questions**
  https://github.com/CPqD/ofsoftswitch13/wiki/Frequently-Asked-Questions