The Road to BOFUSS: The Basic OpenFlow User-space Software Switch

Eder Leão Fernandes[1], Elisa Rojas[2], Joaquin Alvarez-Horcajo[2], Zoltàn Lajos Kis[3], Davide Sanvito[4], Nicola Bonelli[5], Carmelo Cascone[6], and Christian Esteve Rothenberg[7]

[1] Queen Mary University of London, UK
e.leao@qmul.ac.uk
[2] University of Alcala, Spain
{elisa.rojas,j.alvarez}@uah.es
[3] Ericsson, Hungary
zoltan.lajos.kis@ericsson.com
[4] Politecnico di Milano, Italy
davide.sanvito@polimi.it
[5] University of Pisa, Italy
nicola@pfq.io
[6] Open Networking Foundation
carmelo@opennetworking.org
[7] INTRIG, University of Campinas (UNICAMP), Brazil
chesteve@dca.fee.unicamp.br

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

The design and implementation of software for virtual switches is typically complex, requiring from developers knowledge of low level networking details. Even though it is hard to escape the intricate nature of software switches, BOFUSS main focus is simplicity. As such, the design and implementation of components and features from the OpenFlow specification seek for easiness to understand and modify.

The OpenFlow specification does not stipulate data structures and algorithms to implement the pipeline of the switches that support the protocol. As long as the implementation follows the described behavior, there is freedom to define the structure of components. In the design of BOFUSS, whenever possible, the most elementary approach is chosen. Frequently, the straightforward solution is not the most efficient, but exchanging performance for simplicity is a trade-off worth paying, especially when fast prototyping in support of research is prioritized.

We now discuss the structure and organization of BOFUSS, depicted in Figure 1, and how it implements the OpenFlow pipeline. The details presented here aim to be an introduction and 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.

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.

Packet Parser. A Pipeline packet that comes from the switch ports has the header fields extracted by the Packet Parser first. The parsing is automated by the Netbee library [12]. Netbee uses a NetPDL [13] 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. They are the initial part of the OpenFlow pipeline. The fields of a packet's header, parsed by the Packet Parser, are matched against the flows installed in the Flow Tables of the software switch. Matched packets are subject to a set of instructions that may include actions over the packet, e.g., setting one of the fields, or further processing by another table of the Pipeline. The software switch default behavior is to drop any packet that does not match a

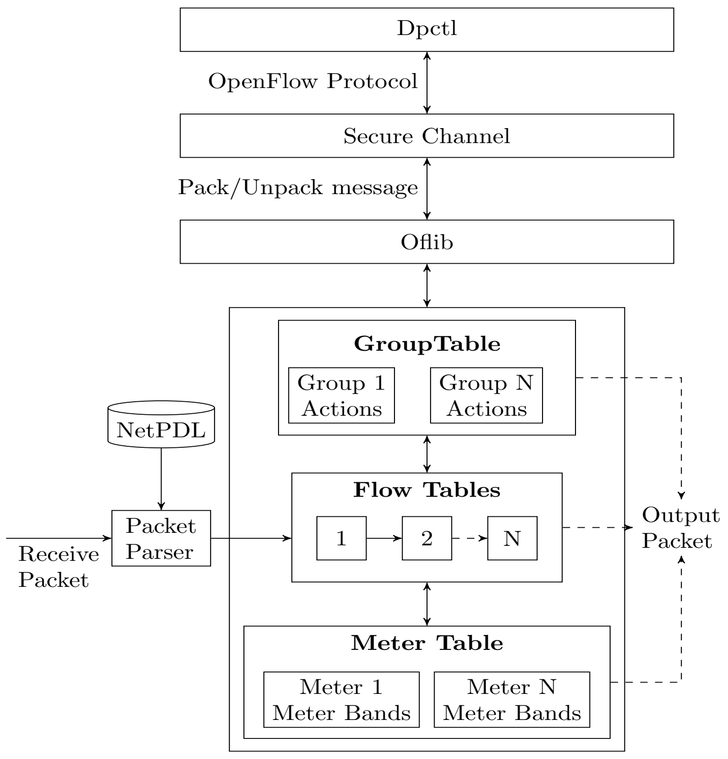

Fig. 1: Overview of the architecture from BOFUSS

flow. The current version of the software switch defines a number of 64 tables in the Pipeline, however, that value can be easily changed to accommodate more.

In BOFUSS, the Flow Tables perform matching in the simplest possible form. Flows are stored in priority order in a linked list. Thus, finding a matching entry has $\mathcal{O}(n)$ complexity. Flow Tables also maintain a list with references to flow entries with hard and idle timeouts, enabling faster checks of expired flows.

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. Actions in a group of the type Select are picked by a simple Round-Robin algorithm. Entries from the Group Table are stored in a hash map for $\mathcal{O}(1)$ retrieval.

Meter Table. Metering was introduced in OpenFlow 1.2 and it gives the possibility to perform Quality of Service (QoS) in 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. A basic Token Bucket algorithm is used to measure the per flow rate and decide if the Meter instruction should be applied or not.

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

Dptcl. The switch includes a command line tool to perform simple monitoring and administration tasks. With Dpctl one can modify and check the current state of switches. A few example of possible tasks: add new flows, retrieve current flow statistics and query the state of ports.

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

OpenState Extension: BEhavioural BAsed forwarding (BEBA) [15] 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 [16], and packet generation, based on InSPired (InSP) switches [17].

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 [18] 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 [19] 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.

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 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 [20]. 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 custom pcap libraries.

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.

Finally, to tackle the parallelism of the multicore architecture, the PFQ [21] framework has been adopted. 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.

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 [22].

3.2 AOSS: OpenFlow hybrid switch

AOSS [23] 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.

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 [24], 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.

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. 2 reflects these functional changes.

Regarding AOSS implementation, two functional changes and two new functions are defined, as defined in Fig. 3. 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.

3.3 OnLife: Deploying the CORD project in a national operator

OnLife [25] is a deployment of the CORD project [26] in Telefonica's[8] 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 clean network deployment from scratch, with no legacy protocols (e.g. allowing only IPv6 and avoiding IPv4).

[8] Main Spanish telecommunications provider

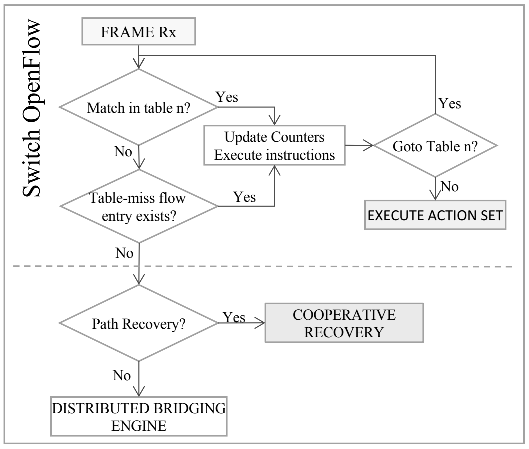

Fig. 2: AOSS's Frame Processing [23]

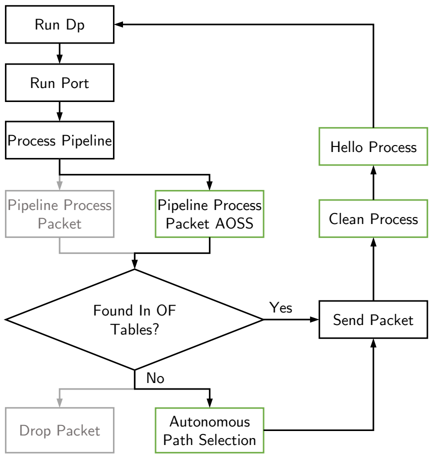

Fig. 3: AOSS's Functional Flow Chart

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 [27]) require IEEE 802.1ad QinQ tunneling [28] 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 [29].

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.

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 Table 1.

4.1 Research implementations or evaluations

Three research implementations have already been introduced in the use cases, namely **OpenState** [16], **InSP** [17] and **AOSS** [23]. All of them envision alternative architectures for SDN in which network switches recover part of the intelligence of the network and, accordingly, they leverage BOFUSS thanks to its easily modifiable pipeline.

Also based on pipeline modifications, **Open Packet Processor (OPP)** [30] enhances the approach of OpenState to support extended Finite State Machines, which broadens the potential functionality of the data plane. **BPFabric** [31,32] 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 [33], which leverages the `select` group of BOFUSS. **RouteFlow** [37] is a pioneering architectural proposal to deliver flexible (virtual) IP routing services over OpenFlow networks [67] (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** [34] 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** [35] modifies BOFUSS to provide scheduled network updates, to avoid potential problems, such as communication loops or blackholes. Finally, **REV** [36] designs a new security primitive for SDN, specifically aimed to prevent rule modification attacks.

Table 1: Classification of works that leverage BOFUSS

Properties Article	Description	Type	Why?
OpenState [16]	OpenFlow extension for stateful applications	Research implementation	Pipeline modification
InSP [17]	API to define in-switch packet generation operations	Research implementation	Pipeline modification
AOSS [23]	Stateful (hybrid) SDN switch	Research implementation	Pipeline modification
OPP [30]	Platform-independent stateful in-network processing	Research implementation	Pipeline modification
BPFabric [31,32]	On-the-fly data plane packet processing pipeline and direct manipulation of network state	Research implementation	Pipeline modification
Fast switchover/failover [33]	New switchover method based on active/active mode (select group)	Research implementation	Pipeline modification
FlowConvertor [34]	Algorithm that provides portability across switch models	Research implementation	Pipeline modification
Chronus [35]	Scheduled consistent network updates	Research implementation	Pipeline modification
REV [36]	New security primitive for SDN	Research implementation	Pipeline modification
RouteFlow [37]	OpenFlow 1.x Dataplane for virtual routing services	Research implementation	OpenFlow version interoperability and Group Tables
TCP connection handover [38]	New method of TCP connection handover in SDN	Research implementation	Modification of OpenFlow 1.3
Facilitating ICN with SDN [39]	Leveraging SDN for ICN scenarios	Research implementation	Extension of OpenFlow
ÆtherFlow [40]	Application of SDN principles to wireless networks	Research implementation	Extension of OpenFlow
CrossFlow [41],[42]	Application of SDN principles to wireless networks	Research implementation	Extension of OpenFlow
Media Independent Management [43]	Dynamic link information acquisition to optimize networks	Research implementation	Extension of OpenFlow
Automatic failure recovery [44]	Proxy between SDN controller and switches to handle failures	Research implementation	Reuses oflib from ofsoftswitch13
OFSwitch13 [45]	Module to enhance the ns-3 simulator with OpenFlow 1.3	Research implementation	Reuses ofsoftswitch13
Time4 [46]	Approach for network updates (adopted in OpenFlow 1.5)	Research implementation	`Bundle` feature
OFLoad [47]	OF-Based Dynamic Load Balancing for data center networks	Research implementation	OpenFlow `group` option
Blind Packet Forwarding in hierarchical architecture [48]	Implementation of the extended BPF	Research implementation	N/D
GPON SDN Switch [49]	GPON based OpenFlow-enabled SDN virtual switch	Research implementation	Part of the architecture
Traffic classification with stateful SDN [50]	Traffic classification in the data plane to offload the control plane	Research implementation	Leverages OpenState [16] and OPP [30]
Traffic classification and control with stateful SDN [51]	Traffic classification in the data plane to offload the control plane	Research implementation	Leverages OpenState [16]
SPIDER [18]	OpenFlow-like pipeline design for failure detection and fast reroute of traffic flows	Research implementation	Leverages OpenState [16]
StateSec [19]	In-switch processing capabilities to detect and mitigate DDoS attacks	Research implementation	Leverages OpenState [16]
Load balancers evaluation [52]	Evaluation of different load balancer apps	Research evaluation	Leverages OpenState [16]
Recovery of multiple failures in SDN [53]	Comparison of OpenState and OpenFlow in multiple-failure scenarios	Research evaluation	Leverages OpenState [16]
UnifyCore [54]	Mobile architecture implementation in which ofsoftswitch13 is leveraged as a fordwarder	PoC implementation	MAC tunneling
ADN [55]	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 [56]	TCP connection handover mechanism implemented in SDN	PoC implementation	Data plane programmability
Multiple Auxiliary TCP/UDP Connections in SDN [57]	Analysis and implementation of multiple connections in SDN	PoC implementation	Extension of OFSwitch13 [45]
State-based security protection mechanisms in SDN [58]	Demonstration of the SDN Configuration (CFG) protection	PoC implementation	Leverages OpenState [16]
Advanced network functions [59]	Stateful data-plane network functions	PoC implementation	Leverages OPP [30]
PathMon [60]	Granular traffic monitoring	PoC implementation	N/D
QoT Estimator in SDN-Controlled ROADM networks [61]	Implementation of a QoT estimator in a simulated optical network	PoC implementation	N/D
OPEN PON [62]	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 [63]	Analysis and implementation of stochastic routing in SDN	PoC implementation (MSc Thesis)	`Select` function of `Group` feature
OpenFlow forwarders [64]	Routing granularity in OpenFlow 1.0 and 1.3	SDN switch comparative	N/A
Open source SDN [65]	Performance of open source SDN virtual switches	SDN switch comparative	N/A
Visual system to learn OF [66]	A visual system to support learning of OpenFlow-based networks	Teaching resource	N/D

N/A means *not applicable*.
N/D means *not defined*.

In the specific case of enhancements of OpenFlow, an extension of OpenFlow 1.3 thanks to BOFUSS is introduced in [38], 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 [39] to provide an optimal parsing of packets in the context of Information-Centric Networking (ICN). Both **ÆtherFlow** [40] and **CrossFlow** [41] 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 [43], 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 [44] reuses the `oflib` library. **OFSwitch13** [45] reuses the whole code of BOFUSS to incorporate the support of OpenFlow 1.3 in the network simulator ns-3. **Time4** [46] reuses the `bundle` feature to implement an approach for network updates (actually adopted in OpenFlow 1.5). **OFLoad** [47] 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 [49].

Finally, several research ideas leverage OpenState and, thus, BOFUSS. The first two were already mentioned previously: **SPIDER** [18] and **StateSec** [19], both examples of stateful applications aimed to provide enhanced network resiliency and monitoring, respectively. Also, **traffic classificators** based on OpenState are also presented in [50] and [51]. Additionally, an evaluation of SDN load balancing implementations is performed in [52], and authors in [53] 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** [54] is an integrated mobile network architecture, based on OpenFlow but leveraging legacy infrastructure. They evaluate the MAC tunneling implemented in BOFUSS with `iperf`. **ADN** [55] describes an architecture that provides QoS based on application flow information, and they chose BOFUSS because it fully supports OpenFlow 1.3. Authors in [56] implemented a novel TCP connection handover mechanism with BOFUSS, aimed to provide transparency 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 [57], while authors in [58] 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 [59].

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** [60], which provides granular traffic monitoring. Another one is a QoT estimator for ROADM networks implemented and evaluated in [61].

Finally, two MSc. Thesis have also be developed based on BOFUSS. The first one is **OPEN PON** [62], 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 [63] 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 [64], and an experimental analysis of different pieces of software in an SDN open source environment [65]. 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 [66], 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* [68] project. The comparison will be performed via two tests:

1. Individual benchmarking of the three switches via iPerf [69]
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[9].

[9] A comparison of OVS with other software switches, but without including BOFUSS, is provided in [70].

Switch	$\overline{x}$	σ
OVS	51,413 Gbps	2,6784
Enhanced BOFUSS	1,184 Gbps	$3,945 * 10^{-3}$
Original BOFUSS	0,186 Gbps	$6,86 * 10^{-5}$

Table 2: Throughput of the three individual types of software switches, measured with iPerf

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 [69]. Our hardware infrastructure consisted of 1 computer powered by Intel(R) Core(TM) i7 processors (3,4 GHz) 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 the average throughput and standard deviation.

The results are shown in Table 2. Although OVS outperforms BOFUSS, it is important to notice how the enhanced switch surpass 1 Gbps,a result considered a reasonable throughput for most common networking scenarios.

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 [71,72,73], 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. 4.

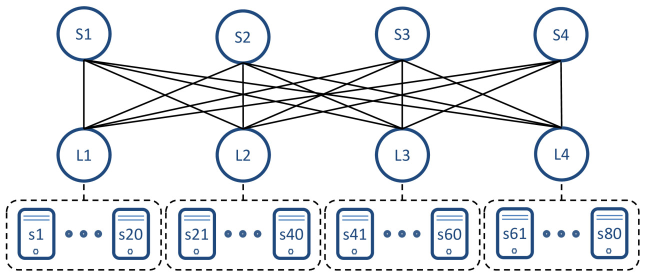

Fig. 4: Spine-Leaf 4-4-20 evaluation topology [74]

To emulate data center-like traffic, we developed a customized traffic generator [74]. This generator implements two different flow size distributions, namely

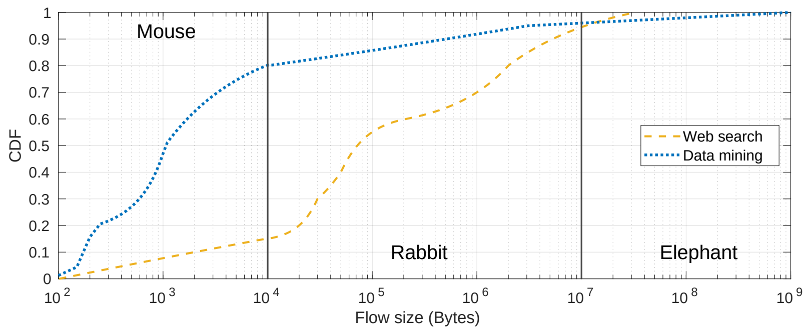

Fig. 5: Flow size distributions [74]

Table 3: Experimental setup of the data center scenarios

Parameter	Value
Network topology	Spine-Leaf (4 - 4)[71]
Servers per leaf switch	20
Flow distribution	Random inter-leaf
Flow size distributions	Web search[75] & Data mining [76]
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 [75,76]. Figure 5 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 [76]. 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 3 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. 6 and

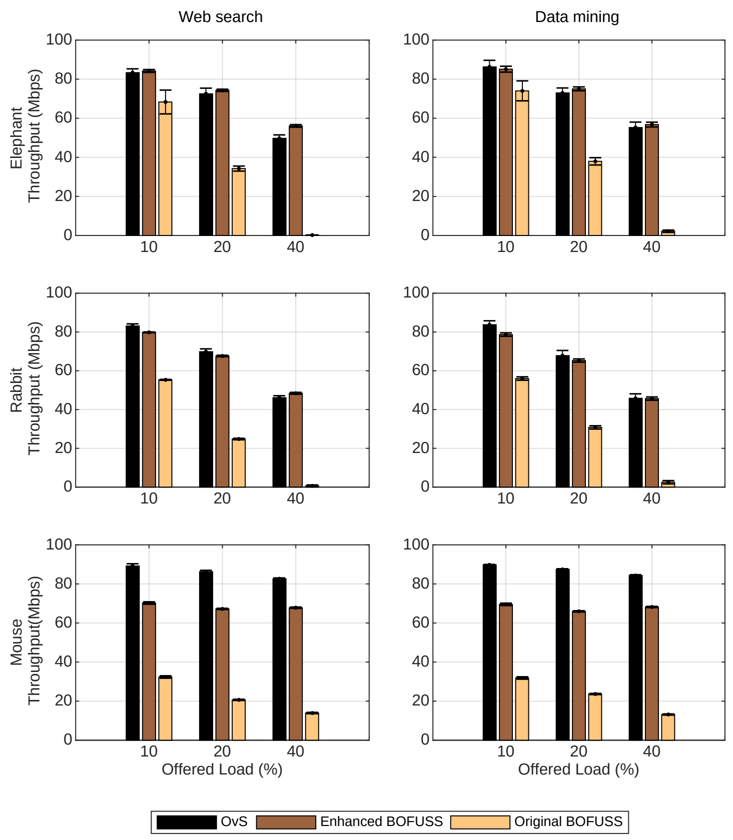

Fig. 6: Throughput in the Spine-Leaf topology for each switch type

Fig. 7, respectively[10]. 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 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.

[10] Raw evaluation data can be found at [77].

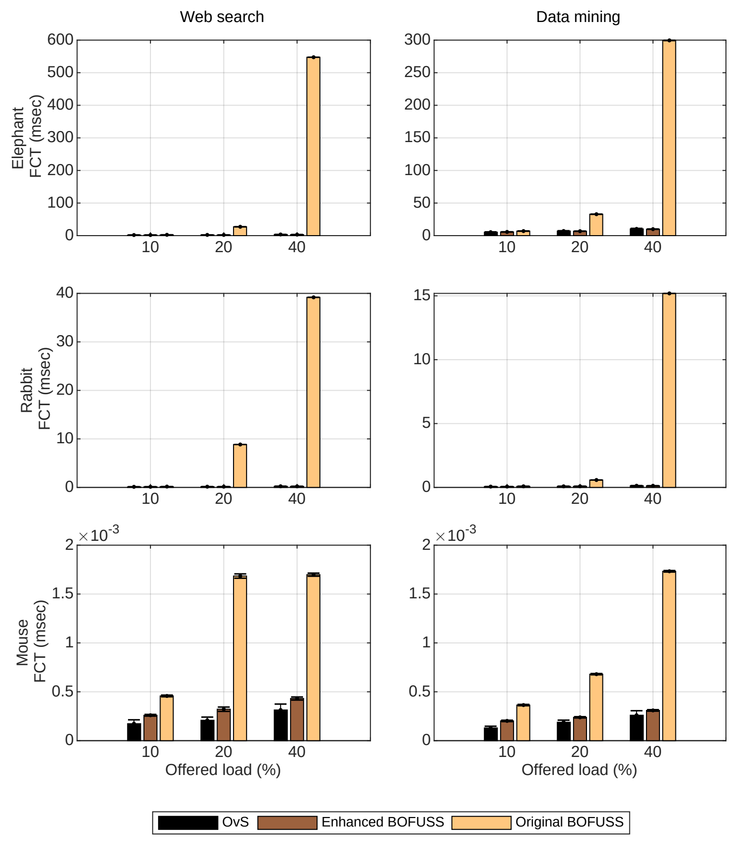

Fig. 7: Flow Completion Time in the Spine-Leaf topology for each switch type

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 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 [78] 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].

Acknowledgements

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.

References

1. D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.
2. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: 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, and M. Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley,CA,USA: USENIX Association, 2015, pp. 117–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789779

4. E. L. Fernandes and C. E. Rothenberg, “OpenFlow 1.3 software switch,” *Salao de Ferramentas do XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos SBRC*, pp. 1021–1028, 2014.
5. “OpenFlow 1.1 Software Switch.” [Online]. Available: https://github.com/TrafficLab/of11softswitch
6. “Stanford OpenFlow Reference Switch repository.” [Online]. Available: http://yuba.stanford.edu/git/gitweb.cgi?p=openflow.git;a=summary
7. “OpenFlow 1.3 switch - CPqD/ofsoftswitch13.” [Online]. Available: https://github.com/CPqD/ofsoftswitch13
8. “OpenFlow Python Switch repositorrty.” [Online]. Available: https://github.com/floodlight/oftest
9. “Open vSwitch.” [Online]. Available: http://openvswitch.org/
10. “OpenFlow Software Switch 1.1 announcement.” [Online]. Available: https://mailman.stanford.edu/pipermail/openflow-discuss/2011-May/002183.html
11. “OpenFlow 1.2 Toolkit announcement.” [Online]. Available: https://mailman.stanford.edu/pipermail/openflow-discuss/2012-July/003479.html
12. “NetBee.” [Online]. Available: https://github.com/netgroup-polito/netbee
13. F. Risso and M. Baldi, “NetPDL: an extensible XML-based language for packet header description,” *Computer Networks*, vol. 50, no. 5, pp. 688–706, 2006.
14. R. R. Denicol, E. L. Fernandes, C. E. Rothenberg, and Z. L. Kis, “On IPv6 support in OpenFlow via flexible match structures,” *OFELIA/CHANGE Summer School*, 2011.
15. “BEBA Behavioral Based Forwarding.” [Online]. Available: http://http://www.beba-project.eu/
16. G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, apr 2014. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602211
17. R. Bifulco, J. Boite, M. Bouet, and F. Schneider, “Improving SDN with InSPired Switches,” in *Proceedings of the Symposium on SDN Research*, ser. SOSR ’16. New York,NY,USA: ACM, 2016, pp. 11:1–11:12. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890962
18. C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sans, “Fast failure detection and recovery in SDN with stateful data plane,” *International Journal of Network Management*, vol. 27, no. 2, pp. e1957–n/a, 2017, e1957 nem.1957. [Online]. Available: http://dx.doi.org/10.1002/nem.1957
19. J. Boite, P. A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, “Statesec: Stateful monitoring for DDoS protection in software defined networks,” in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–9.
20. “TCPDUMP/LIBPCAP public repository.” [Online]. Available: https://www.tcpdump.org/
21. N. Bonelli, S. Giordano, and G. Procissi, “Network Traffic Processing With PFQ,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1819–1833, June 2016.
22. N. Bonelli, G. Procissi, D. Sanvito, and R. Bifulco, “The acceleration of OfSoftSwitch,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2017, pp. 1–6.
23. J. Alvarez-Horcajo, I. Martinez-Yelmo, E. Rojas, J. A. Carral, and D. Lopez-Pajares, “New cooperative mechanisms for software defined networks based on hybrid switches,” *Transactions on Emerging Telecommunications Technologies*,

pp. e3150–n/a, 2017, e3150 ett.3150. [Online]. Available: http://dx.doi.org/10.1002/ett.3150

24. E. Rojas, G. Ibanez, J. M. Gimenez-Guzman, J. A. Carral, A. Garcia-Martinez, I. Martinez-Yelmo, and J. M. Arco, "All-Path bridging: Path exploration protocols for data center and campus networks," *Computer Networks*, vol. 79, no. Supplement C, pp. 120 – 132, 2015.
25. R. S. Montero, E. Rojas, A. A. Carrillo, and I. M. Llorente, "Extending the Cloud to the Network Edge," *Computer*, vol. 50, no. 4, pp. 91–95, 2017.
26. 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 Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.
27. P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open,Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York,NY,USA: ACM, 2014, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620744
28. "802.1ad - Provider Bridges." [Online]. Available: http://www.ieee802.org/1/pages/802.1ad.html
29. "Master in NFV and SDN for 5G Networks. UC3M." [Online]. Available: https://www.uc3m.es/master/NFV-SDN-5g-networks
30. G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone, "Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing," *ArXiv e-prints*, may 2016. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv160501977B
31. S. Jouet, R. Cziva, and D. P. Pezaros, "Arbitrary packet matching in OpenFlow," in *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, July 2015, pp. 1–6.
32. S. Jouet and D. P. Pezaros, "BPFabric: Data Plane Programmability for Software Defined Networks," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '17. Piscataway,NJ,USA: IEEE Press, 2017, pp. 38–48. [Online]. Available: https://doi.org/10.1109/ANCS.2017.14
33. K. Nguyen, Q. T. Minh, and S. Yamada, "Novel fast switchover on OpenFlow switch," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan 2014, pp. 543–544.
34. H. Pan, G. Xie, Z. Li, P. He, and L. Mathy, "FlowConvertor: Enabling portability of SDN applications," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.
35. J. Zheng, G. Chen, S. Schmid, H. Dai, J. Wu, and Q. Ni, "Scheduling Congestion- and Loop-Free Network Update in Timed SDNs," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2542–2552, Nov 2017.
36. P. Zhang, "Towards rule enforcement verification for software defined networks," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.
37. A. Vidal12, F. Verdi, E. L. Fernandes, C. E. Rothenberg, and M. R. Salvador, "Building upon RouteFlow: a SDN development experience."
38. A. Binder, T. Boros, and I. Kotuliak, *A SDN Based Method of TCP Connection Handover*. Cham: Springer International Publishing, 2015, pp. 13–19.
39. P. Zuraniewski, N. van Adrichem, D. Ravesteijn, W. IJntema, C. Papadopoulos, and C. Fan, "Facilitating ICN Deployment with an Extended Openflow Protocol,"

in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ser. ICN '17. New York,NY,USA: ACM, 2017, pp. 123–133. [Online]. Available: http://doi.acm.org/10.1145/3125719.3125729

40. M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton, "ÆtherFlow: Principled Wireless Support in SDN," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, Nov 2015, pp. 432–437.
41. P. Shome, M. Yan, S. M. Najafabad, N. Mastronarde, and 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)*, Nov 2015, pp. 37–39.
42. P. Shome, J. Modares, N. Mastronarde, and A. Sprintson, "Enabling Dynamic Reconfigurability of SDRs Using SDN Principles," in *Ad Hoc Networks*. Springer, 2017, pp. 369–381.
43. C. Guimares, D. Corujo, and R. L. Aguiar, "Enhancing openflow with Media Independent Management capabilities," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 2995–3000.
44. M. Kuźniar, P. Perešíni, N. Vasić, M. Canini, and 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.
45. L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "OFSwitch13: Enhancing Ns-3 with OpenFlow 1.3 Support," in *Proceedings of the Workshop on Ns-3*, ser. WNS3 '16. New York,NY,USA: ACM, 2016, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/2915371.2915381
46. T. Mizrahi and Y. Moses, "Time4: Time for SDN," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 433–446, Sept 2016.
47. R. Trestian, K. Katrinis, and G. M. Muntean, "OFLoad: An OpenFlow-Based Dynamic Load Balancing Strategy for Datacenter Networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 792–803, Dec 2017.
48. I. Simsek, Y. I. Jerschow, M. Becke, and 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)*, Dec 2014, pp. 1–5.
49. S. S. W. Lee, K. Y. Li, and M. S. Wu, "Design and Implementation of a GPON-Based Virtual OpenFlow-Enabled SDN Switch," *Journal of Lightwave Technology*, vol. 34, no. 10, pp. 2552–2561, May 2016.
50. D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful SDN data planes," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–4.
51. A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, and T. Zhang, "On-the-fly traffic classification and control with a stateful SDN approach," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.
52. W. J. A. Silva, K. L. Dias, and D. F. H. Sadok, "A performance evaluation of Software Defined Networking load balancers implementations," in *2017 International Conference on Information Networking (ICOIN)*, Jan 2017, pp. 132–137.
53. M. S. M. Zahid, B. Isyaku, and 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)*, Oct 2017, pp. 1178–1183.
54. M. Nagy, I. Kotuliak, J. Skalny, M. Kalcok, and T. Hirjak, *Integrating Mobile OpenFlow Based Network Architecture with Legacy Infrastructure*. Cham: Springer International Publishing, 2015, pp. 40–49.

55. F. S. Tegueu, S. Abdellatif, T. Villemur, P. Berthou, and T. Plesse, “Towards application driven networking,” in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2016, pp. 1–6.
56. W. Fan and D. Fernandez, “A novel SDN based stealthy TCP connection handover mechanism for hybrid honeypot systems,” in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–9.
57. H. Yang, C. Zhang, and G. Riley, “Support Multiple Auxiliary TCP/UDP Connections in SDN Simulations Based on Ns-3,” in *Proceedings of the Workshop on Ns-3*, ser. WNS3 ’17. New York,NY,USA: ACM, 2017, pp. 24–30. [Online]. Available: http://doi.acm.org/10.1145/3067665.3067670
58. T. Arumugam and 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)*, Nov 2017, pp. 123–125.
59. M. Bonola, R. Bifulco, L. Petrucci, S. Pontarelli, A. Tulumello, and G. Bianchi, “Demo: Implementing advanced network functions with stateful programmable data planes,” in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2017, pp. 1–2.
60. M.-H. Wang, S.-Y. Wu, L.-H. Yen, and C.-C. Tseng, “PathMon: Path-specific traffic monitoring in OpenFlow-enabled networks,” in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, July 2016, pp. 775–780.
61. A. A. Díaz-Montiel, J. Yu, W. Mo, Y. Li, D. C. Kilper, and M. Ruffini, “Performance analysis of QoT estimator in SDN-controlled ROADM networks,” in *2018 International Conference on Optical Network Design and Modeling (ONDM)*, May 2018, pp. 142–147.
62. M. D. M. Silva and N. Filipe, “OPEN PON: Seamless integration between 5G core and optical access networks,” Master’s thesis, Universitat Politècnica de Catalunya, 2016.
63. K. Shahmir Shourmasti, “Stochastic Switching Using OpenFlow,” Master’s thesis, Institutt for telematikk, 2013.
64. V. Šulák, P. Helebrandt, and 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)*, Nov 2016, pp. 236–241.
65. K. Tantayakul, R. Dhaou, B. Paillassa, and 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)*, June 2017, pp. 334–337.
66. H. Fujita, Y. Taniguchi, and 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)*, Oct 2017, pp. 1–3.
67. C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and 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*, ser. HotSDN ’12. New York, NY, USA: ACM, 2012, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342445
68. “BEBA Software Switch.” [Online]. Available: https://github.com/ccascone/beba-switch
69. “iPerf - The TCP, UDP and SCTP network bandwidth measurement tool.” [Online]. Available: https://iperf.fr/

70. V. Fang, T. Lvai, S. Han, S. Ratnasamy, B. Raghavan, and J. Sherry, "Evaluating Software Switches: Hard or Hopeless?" EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2018-136, Oct 2018. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-136.html
71. M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, and e. a. Yadav, "CONGA:Distributed Congestion-aware Load Balancing for Datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, aug 2014.
72. K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto:Edge-based Load Balancing for Fast Datacenter Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, aug 2015.
73. M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric:Minimal Near-optimal Datacenter Transport," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, aug 2013.
74. J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, and I. Martinez-Yelmo, "TCP-path: Improving load balance by network exploration," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Sep. 2017, pp. 1–6.
75. M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 63–74, aug 2010.
76. A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2:a scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.
77. J. Alvarez-Horcajo, "BOFUSS raw evaluation data." [Online]. Available: https://github.com/gistnetserv-uah/GIST-DataRepo/tree/master/BOFUSS-spine_leaf-190120
78. P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

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