LoCoSDN: A Local Controller for Operation of OF Switches in non-SDN Networks

Mark Schmidt, Frederik Hauser, Bastian Germann, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {mark-thomas.schmidt,frederik.hauser,menth}@uni-tuebingen.de, bastian.germann@student.uni-tuebingen.de

***Abstract*—Hybrid OpenFlow (OF) switches can be operated as legacy switches with a local control plane or as OF switches that are controlled by an SDN controller. As a first thought, hybrid switches are the first choice for network administrators as they facilitate the transition from a traditionally operated towards a software-defined network. However, we encountered several limitations as most of these switches are based on a legacy hardware platform that is augmented with a limited set of OF features. As a consequence, some packet processing pipelines of hybrid switches filter out packets before they pass the OF part. To avoid these issues, we developed LoCoSDN, an SDN architecture that imitates the hybrid switch behavior on OF-only devices, i.e., they can be operated in traditional networks and without any special knowledge about SDN. LoCoSDN contains a Cisco-like CLI and configuration format so that configurations from existing devices can be imported. It differentiates between a startup and running configuration, and provides reconfigurability during operation and support for an additional SDN controller. We implemented LoCoSDN as a lightweight SDN controller for the Ryu SDN controller framework that can be run on a single-board computer.**

I. Introduction

The current landscape of hardware-based network switches includes three different architectures that are depicted in Figure 1. Traditional network switches (1) consist of a data plane and a control plane. The data plane is responsible for fast packet processing and programmed by the control plane. It contains algorithms for path computations and implements various network mechanisms and protocols along with user interfaces that allow network administrators to configure the device. Software-defined networking (SDN) splits the strong binding between data and control plane. OpenFlow (OF) is the most widespread standard for SDN that defines an architecture and a communication protocol between SDN switches and the SDN controller. OF-only switches (2) only contain a data plane and require an SDN controller that provides all functionality of the former control plane. Protocols and mechanisms need to be implemented in software running on the SDN controller. Hybrid OF switches (3) contain the control plane of a legacy switch with an interface to an OF controller. Forwarding decisions are either made by the switch's legacy control plane or by an external SDN controller. Current hybrid OF switches are mostly legacy hardware switches that are augmented with a limited set of OF features. Therefore, hardware parts of the switch such as TCAM, ASIC, and switch memory are used to implement OF support through firmware updates.

This work was supported by the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

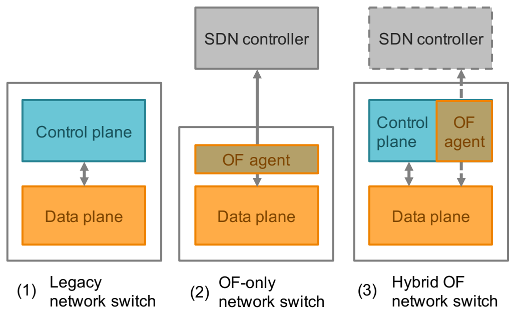


Fig. 1: Three different architectures for network switches. The legacy switch combines a control and data plane on a single device. The OF-only switch requires an SDN controller as substitute for the local data plane. The hybrid OpenFlow switch combines a local control plane with an external SDN controller.

Nowadays, only a few companies are ready to apply SDN control to their networks. Others want to go this step only in the future. When they invest now into new hardware, they have two choices so that they can reuse the equipment in the SDN network: hybrid OF switches or OF-only switches. In comparison to OF-only switches, hybrid OF switches bring many benefits: legacy control plane functions such as L2 switching, L3 routing, and VLAN tagging as well as legacy user interfaces are further on usable. However, we encountered several limitations that are caused by the difficulties in distinguishing between a local control plane and OF control. In particular, we observed packet processing pipelines of hybrid OF switches that filter out packets so that they did not reach the SDN controller.

As an alternative for current hybrid OF switch architectures, we propose LoCoSDN. It imitates the hybrid switch behavior by coupling an OF-only switch with a local SDN controller that can be run on cheap commodity hardware platforms, e.g., a Raspberry Pi. In contrast to hybrid OF architectures, all parts and interfaces of LoCoSDN are open-source. LoCoSDN offers the legacy functions and user interfaces of a legacy switch so that network administrators do not have to know SDN specifics as long as the hybrid switch is operated in the legacy mode. If LoCoSDN should be used within an SDN infrastructure, it may be configured to connect to an external SDN controller and pass-through the OF communication to the switch.

The remainder of the paper is structured as follows. Section II provides an overview of related work. In Section III, we outline the basic idea of our concept. The internal design of the local SDN controller as central part of LoCoSDN is described in Section IV. The prototypical implementation and a functional validation is described in Section V. Section VI concludes this work.

II. Related Work

In the following, we describe the characteristics and architecture of legacy network switches, introduce OF-only, hybrid, and whitebox OF switches, and discuss related approaches for bringing hybrid-like behavior to OF-only switches.

A. *Legacy Network Switches*

Legacy network switches offer many control plane functions on different layers. Protocol-related functionalities include ARP, MPLS, Q-in-Q, and ICMP, additional features include access control lists (ACLs), mechanisms for authentication and authorization, and vendor-specific interfaces to external systems.

This large variety of control plane functions requires extensive and complex configuration. Therefore, legacy network switches offer user interfaces and network management protocols. Most legacy network switches include two types of user interfaces: web interfaces and command line interfaces (CLIs). Web interfaces are graphical user interfaces that can be accessed using a browser. CLIs represent simpler interfaces where network administrators connect to a switch either over a serial, Telnet, or secure shell (SSH) connection. Then, ASCII-based human-to-machine languages are either used in an interactive console prompt or for manually typing scripts or configurations. Although syntax can be vendor-specific, Cisco-like CLIs emerged as de-facto standard for legacy network devices from various manufacturers. They are even used in software tools [1] and operating systems [2] for interactive configuration. Network management protocols such as SNMP [3] or NETCONF [4] aim at network infrastructures with many devices that need to be managed and operated.

B. *OF Switches*

The current landscape of OF switches can be divided into OF-only and hybrid switch architectures. We classify whitebox switches as either OF-only or hybrid OF switches, depending on the particular form.

1) OF-Only Switches: The minimal architecture for an OF switch strictly follows the OF switch specification [5] and does not contain a legacy control plane. Instead, it has a microprocessor running a firmware that includes an OF agent as interface between the OF-only switch and an SDN controller. OF-only switch architectures are mostly found in software switches: Lagopus [6] or LINC-Switch [7] are examples for OF-only switches. However, there is only a limited number of OF-only hardware switches such as the NEC ProgrammableFlow series [8], the NoviFlow NOVISWITCHes [9], or the experimental SDN switch platform Zodiac FX [10].

2) Hybrid OF Switches: This type of OF switch architecture combines a legacy control plane with an additional option to use an external SDN controller. Figure 2 depicts the internal architecture of a hybrid OF switch. The data plane is programmed by a local control plane (1). It includes an on-board logic (2) implementing basic network functionalities such as L2 switching, L3 routing, or VLAN tagging, and contains interfaces to external systems (3). The OF agent (4) as additional component on the control plane represents the interface to the SDN controller (5).

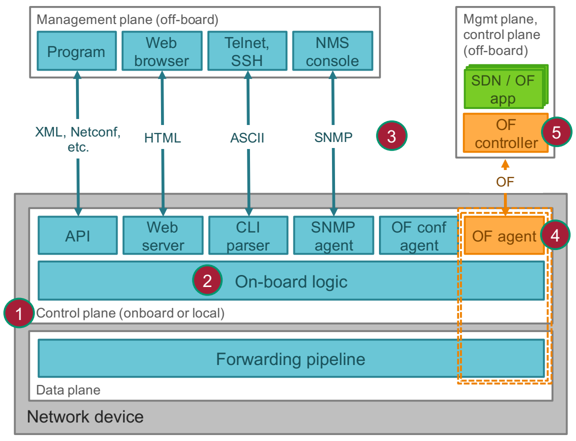


Fig. 2: Internal architecture of a hybrid OF switch consisting of a data and control plane (according to [11]).

The control plane firmware of hardware hybrid OF switches is closed-source and mostly tailored to a particular hardware model. It comes with a basic set of features, while additional features can be activated by purchasing upgrade licenses. In addition, the interface between the control and data plane is based on a proprietary protocol. As a result, neither the operating system nor the functionalities can be extended or exchanged. For virtualized environments, also software hybrid switches like Open vSwitch (OVS) [12] exist.

Still, this concept offers many benefits over OF-only switches. First, control plane functions such as L2 switching or L3 routing need to be implemented on the SDN controller first when OF-only switches are rolled out. Hybrid OF switches allow a step-by-step transition: legacy control plane functions are still available on the local control plane, an SDN controller is not required at this point. However, an SDN controller may introduce new functionalities at a later time. Second, hybrid OF switches enable parallel operation of local control plane functions and of an SDN controller. As an example, ARP functionality for L2 switching may be provided by the local control plane whereas L3 routing is achieved by an SDN controller. This reduces OF control plane traffic and load on the SDN controller as some forwarding decisions are made locally. It also reduces the latency for installing forwarding rules resulting from the communication between SDN switches and the SDN controller which may be physically distant. Last, the hybrid architecture lowers the barrier for network administrators to adopt OF-based SDN because features and well-known user interfaces such as CLIs of legacy switches are still present.

As a consequence of the hybrid architecture, forwarding decisions for incoming data are either made by the local control plane or the SDN controller. That requires a separation mechanism. One approach defines the separation on a per port or per VLAN base. Another approach defines the separation on a per function base. As an example, an SDN controller is used for L3 routing while L2 switching based on MAC addresses still remains on the legacy control plane.

Typical hybrid switches are mostly based on a traditional switch design which are extended with OF functionality. This architecture may lead to several problems. E.g., in a previous work [13], we experienced problems in forwarding EAPoL packets in an 802.1X test infrastructure from a hybrid switch to an SDN controller. The manufacturer's OF documentation [14] for that particular switch model shows a list of incompatibilities when using OF.

3) Whitebox Switches: The architecture is mainly driven by large network operators and enterprises. To lower CAPEX, low-cost original device manufacturers use generic, off-the-shelf switching hardware to build SDN hardware switches as an alternative to legacy, OF-only, or hybrid OF network switches from big vendors. Whitebox switches like the OPEN NETWORKING series from Edgecore [15] contain merchant silicon switching chips, a commodity CPU, and memory. They rely on an either proprietary or open-source network operating system (NOS) [16] which can be installed via the Open Network Install Environment (ONIE) [17]. Two widely used examples are Cumulus Linux [18] and the Facebook Open Switching System (FBOSS) [19]. Cumulus Linux is an example for a heavyweight NOS that requires a powerful computing platform on the network hardware, i.e., bare metal switches that include multi-core CPUs, RAM, and a SSD. It is based on a modified Debian distribution with interfaces to hardware components such as routing tables and switching ASICs. Besides providing an OF interface for complete external control, the switch can be operated in a legacy manner as well. Linux tools such as iproute2 [20] or Quagga [1] can be used to control the data plane just like for hybrid OF switches as discussed before. FBOSS is an example for an open-source and rather lightweight NOS. API libraries such as OpenNSL [21] and OF-DPA [22] run on a microprocessing unit to interface specific switch ASICs.

Figure 3 depicts the difference between the hybrid and OF-only whitebox switch architectures. Hybrid whitebox switch architectures for running NOS such as Cumulus Linux contain legacy network functions as part of their control plane. OF-only whitebox switch architectures solely contain an OF agent as interface to an external SDN controller.

C. Alternative Hybrid OF Switch Architectures

In contrast to OF-only switches, hybrid OF switches contain a local control plane that primarily consists of two functional parts: control plane functionalities and interfaces for monitoring and configuration. In the following, we present

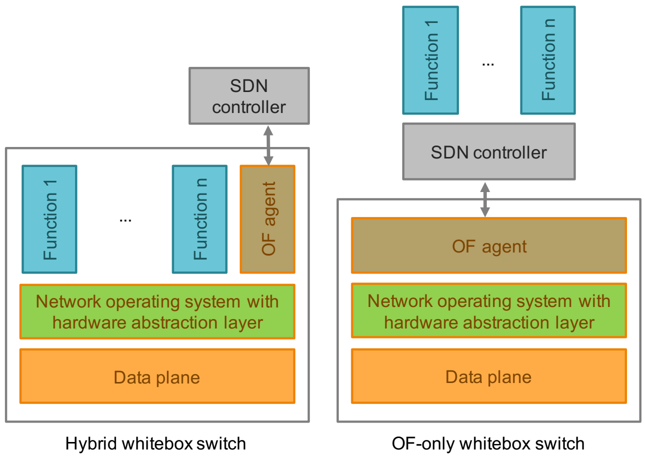


Fig. 3: Comparison of hybrid and OF-only whitebox switch architectures.

related approaches that try to introduce hybrid-like behavior to infrastructures that are built of OF-only switches and SDN controllers by implementing the two functional parts.

1) Legacy Control Plane Functionalities: Valve [23] is an SDN application that runs on top of the Ryu SDN controller framework. It introduces various functionalities from legacy control planes such as VLAN tagging, IPv4/IPv6 ACLs, auto-configuration of ports, and port statistics. Valve is configured within a static YAML file and can be run in parallel with other SDN controllers. Therefore, flow entries installed by Valve are identified by a special and unique OF cookie. Faucet [24], [25] is an improvement of Valve and currently used in various enterprise networks, including the administration network of the Open Networking Foundation. Faucet supports VLAN switching, IPv4/IPv6 routing, ACLs, port mirroring, and policy-based forwarding. In addition, it pushes statistical information to an external database and offers to use Grafana to construct visualization views. Identical to Valve, Faucet uses individual YAML files for configuration. Dragonflow [26] is an SDN controller for the OpenStack Neutron [27] platform. It is also based on the Ryu SDN controller framework and provides several applications that implement legacy control plane functionalities such as L2 switching, L3 routing, and VLAN tagging. However, due to the fixed integration into the OpenStack platform, the configuration of DragonFlow is fully automated and based on the platform's orchestrator.

2) User Interfaces to Control Plane Functions: CLIs represent the common way to configure traditional switches. The following approaches introduce CLIs on SDN controllers to ease the transition for network administrators. SDNsh [28] is an advanced CLI for the OpenDaylight SDN controller platform and the Open Network Operating System. It offers a Python-based CLI depending on a running instance of sdncon, a northbound interface of the two platforms. OpenMUL [29] is an SDN controller that provides a CLI to specify the entries of the flow tables, specify actions, and define OF pipelines. However, both approaches do not come with built-in commands for traditional networking.

III. Architecture of LoCoSDN

In this section, we describe the physical setup of LoCoSDN.

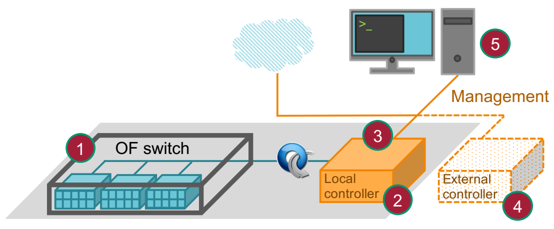


Fig. 4: Physical setup of LoCoSDN consisting of an OF switch, a local controller, a management station, and an optional external controller.

Figure 4 depicts the physical architecture of LoCoSDN. An OF-only switch (1) is directly connected via an Ethernet cable to a cheap single-board computer (SBC) that runs a local SDN controller (2). OF is used as open and extensible southbound protocol on the link between the data (1) and control plane (2). The SBC provides another physical interface (3) to connect to a management network that can consist of additional external SDN controllers (4) or management computers (5).

We chose to use an *OF-only switch* (1) as datapath element. As introduced in Section II, it represents a minimal architecture for an OF switch which only forwards packets on physical ports based on entries in the OF flow table. It has an OF agent on a microprocessor that holds an OF connection with an SDN controller over an Ethernet link. *The LoCoSDN's controller* (2) is the key element of the architecture. It runs on an SBC and realizes the entire control plane functionality of the LoCoSDN architecture. It implements all control plane functionalities such as L2 switching, L3 routing, or VLAN tagging and includes known user interfaces such as a CLI and web interface. LoCoSDN's SBC running the local SDN controller contains another physical Ethernet port. It is used to connect a management network that can host *additional SDN controllers* (4) and *management computers* (5). Additional SDN controllers may be either used as a substitute or in conjunction with LoCoSDN's local SDN controller. External SDN controllers may be responsible for multiple switches and therefore provide a more global view of the network which enables global optimizations and more complex traffic steering. Management computers configure switches and routers either through network management protocols such as SNMP or user interfaces. LoCoSDN's SDN controller offers a Cisco-like CLI and web-based GUI for administration. The size of the management network can vary greatly, ranging from a single management computer up to an infrastructure with numerous distributed SDN controller instances.

IV. The LoCoSDN Controller

In the following, we present the modes, the module system, the storage model, and the configuration model of the LoCoSDN controller in detail.

A. Modes

LoCoSDN supports three modes of operation: local controller, remote controller, and hybrid mode. In *local controller mode*, no external SDN controller is used at all. Instead, all desired network functionalities are either provided by basic SDN applications that are part of the LoCoSDN controller or by additional modules. In *remote controller mode*, LoCoSDN's local SDN controller is not used at all. Instead, all control plane functionalities are provided by an external SDN controller that is attached to the management network. In use cases where a remote controller manages the entire switch, the local SDN controller is not needed, i.e., the switch has to be reconfigured to connect to an external controller and after that, the local controller can be switched off. In *hybrid mode*, data plane control can be either done by the local controller or by an external SDN controller on a per-port base. This mode is useful for scenarios where some ports of the switch are managed by the local controller and some ports are configured by a global controller. Therefore, the network administrator needs to define which physical port of the OF switch belongs to which controller. In this mode, LoCoSDN's local controller acts as a proxy to external SDN controllers and ensures that the SDN controllers can only install flow rules lying in their respective range of authority. Figure 5 depicts an exemplary scenario of the hybrid mode. Several physical ports are controlled by LoCoSDN's local SDN controller (1) where other physical ports are controlled by the external SDN controller (2).

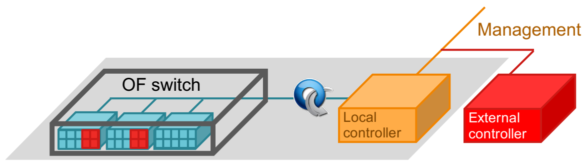


Fig. 5: Multiple physical ports of a LoCoSDN switch are controlled by the local SDN controller, other physical ports are controlled by an external SDN controller.

B. Module System

When operated in local controller or hybrid mode, LoCoSDN's SDN controller provides the control plane functionalities of a legacy network switch. L2 switching, L3 routing, and VLAN tagging are examples for legacy control plane network functions, the AAM as presented in [13] is a novel functionality. LoCoSDN splits those different functions into dedicated modules. Figure 6 depicts the mechanism of LoCoSDN's module system. It consists of three layers: the SDN controller layer (1), the module layer (2), and the configuration layer (3). Each module consists of controller logic that realizes the actual functionality and a parser for the configuration data.

As depicted in Figure 6, LoCoSDN's SDN controller already includes basic modules such as L2 switching, L3 routing, and VLAN tagging depicted in (2)(a). Missing features can be implemented in add-on modules like shown in (2)(b). An example add-on module is the AAM [13].

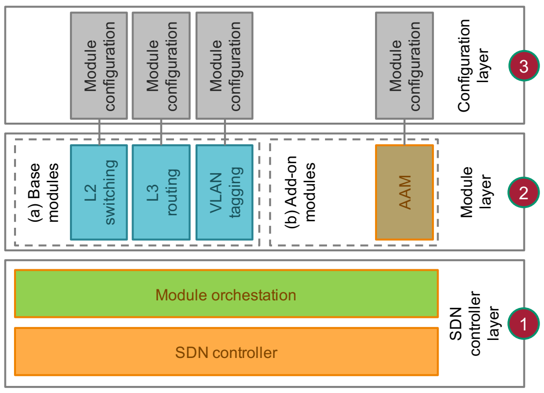


Fig. 6: Simple module mechanism for the LoCoSDN switch consisting of a base module that registers add-on modules and configuration data.

C. *Storage Model*

Figure 7 shows the storage model of LoCoSDN that is split up into two partitions. The *firmware partition* (1) holds a minimal Linux operating system with an SDN controller. Basic control plane functionalities as discussed before are encapsulated in base modules. The firmware partition is bundled as a downloadable image so that upgrading the whole firmware consisting of operating system and SDN controller is an easy task. The *persistent storage partition* (2) contains the module configurations and add-on modules. This partition is not affected by changes in the firmware partition so that the operating system and the SDN controller can be upgraded without losing configuration data or base modules.

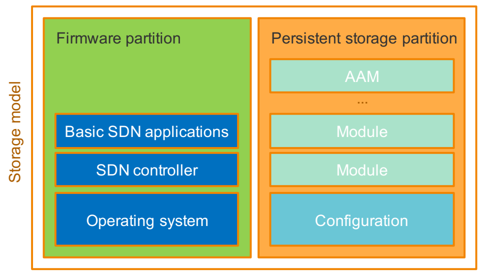


Fig. 7: Storage layout of the LoCoSDN consisting of a firmware partition and a partition for configuration data and additional modules.

D. *Configuration Model*

Figure 8 depicts the internal structure of the local controller. It consists of three main blocks: the configuration front end (1), the configuration back end (2), and the actual SDN controller (3).

The *configuration front end* includes two different user interfaces: a CLI and a web-based GUI. A Cisco-like CLI offers administrators full access to all features of LoCoSDN's local SDN controller, i.e., all functionalities can be configured exactly like on legacy network switches. To enable a soft transition from existing deployments that are based on traditional switches, it allows importing already existing network configuration data in the Cisco format. An adopted Cisco format is used as native configuration format. The web-interface provides a simplified interface for end users.

Both user interfaces communicate via API calls with the *configuration back end*. Like in typical traditional switches or routers, the configuration distinguishes between a startup config and a running config. The startup config represents the persistent configuration that is loaded at system start and then copied to a temporary running config. Changes to the configuration are only applied to the running config and require an explicit write command to be made persistent. The advantage of this mechanism is that the startup config should always be in a working state and in case of a misconfiguration, a reboot returns into a working state.

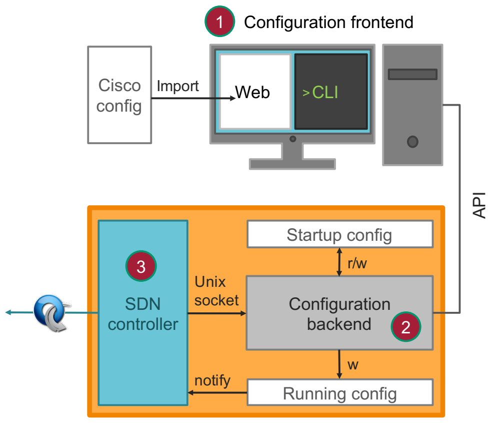


Fig. 8: The local controller consists of a configuration front and back end and an SDN controller communicating with each other.

The actual *SDN controller* realizes the functionality of a traditional L3 switch and configures the SDN switch via OF accordingly. To that end, the controller needs to implement parsers for the different parts of the running config and the corresponding functionality to translate the configuration data into flow rules.

In case of changes to the running config, the controller gets notified to re-read the data. After that, the modified flow rules are applied at the switch. Information from the switch like, e.g., port-up or port-down messages, are forwarded to the configuration back end via a custom API and visualized in the configuration front end.

V. Prototypical Implementation & Functional Validation

In the following, we describe the hardware and software that we used to implement the prototype and describe our prototypical validation.

A. *Prototypical Implementation*

Figure 9 depicts the hardware architecture of the prototypical implementation. The prototype runs on an inexpensive experimental hardware platform and focuses on features rather than performance. We decided to use a Zodiac FX [10] as OF-only switch. It is the result of a crowdfunding campaign [30] initiated in 2015 by Northbound Networks, the current seller of the switch. The Zodiac FX is an open hardware platform that is based on the Atmel ATSAM4E Cortex M4 microprocessor. It offers four 100 Mb/s links and support for OF in version 1.0 and 1.3. The firmware of the Zodiac FX is open source [31] which allowed us to fix a firmware bug in VLAN tagging. We further decided to use the Raspberry Pi 3 [32] microcomputing platform with a quad-core ARM CPU and 1GB RAM as SBC running LoCoSDN's local SDN controller. The first USB port of the Raspberry Pi is used as power supply to the Zodiac FX switch, the other USB port connects a USB NIC for the management network. This platform provides enough hardware resources to run LoCoSDN's local SDN controller with typical network functions such as L2 switching, L3 routing, or VLAN tagging. If more complex and CPU-intense network functions should be provided, the hardware platform might be substituted by a more powerful platform.

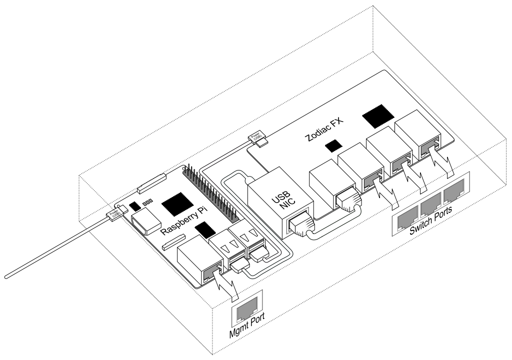
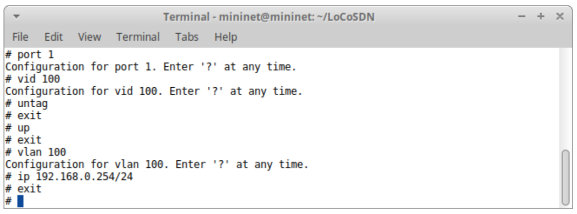


Fig. 9: Assembled hardware platform of LoCoSDN consisting of a Zodiac FX OF-only switch, a Raspberry Pi 3, and a USB NIC.

We chose RASPBIAN [33] as Linux-based operating system for the SBC. We implemented LoCoSDN's local SDN controller using the Ryu SDN controller framework [34]. It is implemented in Python 3 and optimized for rapid prototyping using a lightweight architecture. The implementation of LoCoSDN's local SDN controller encompasses two parts: the network functions and the user interfaces of legacy control planes. On the network function side, we implemented L2 switching, L3 routing, and VLAN tagging. On the user interface side, we implemented a specific northbound interface to communicate with the configuration back end that uses two different methods. The SDN controller reads its configuration data from a file and regularly polls it for updates. A Unix socket signals events from the switch to the configuration back end. The configuration back end and front end are also implemented in Python and communicate with each other directly via API calls. For evaluation purposes, we implemented a fully functional CLI as shown in Figure 10 and a read-only web-interface to display the current configuration and state of the LoCoSDN switch.

B. Functional Validation

To validate the prototypical implementation, we built a simple test setup consisting of three LoCoSDN switches that are connected by Ethernet links like shown in Figure 11. Each



```
# port 1
Configuration for port 1. Enter '?' at any time.
# vid 100
Configuration for vid 100. Enter '?' at any time.
# untag
# exit
# up
# exit
# vlan 100
Configuration for vlan 100. Enter '?' at any time.
# ip 192.168.0.254/24
# exit
# 
```

Fig. 10: Screenshot of LoCoSDN's CLI presenting the VLAN configuration.

LoCoSDN switch has a Raspberry Pi 3 acting as network client attached to it via an Ethernet link. The management port of each LoCoSDN switch is connected to a 5-port legacy switch that also includes a notebook for management. For an initial test, we configure the Client1 and Client2 with an IP address in the subnet 10.0.1.0/24. The two clients can reach each other via the switch. Next, we configure the port towards Client1 with VLAN 1 and the port towards Client2 with VLAN 2. As expected, the two clients cannot reach each other any longer. As a final configuration test, we configure for Client1, Client2, and Client3 an IP address in the IP subnet 10.0.2.0/24 and set their access ports on the switches to VLAN 2. The ports connecting both switches are configured with tagged in VLAN 2. Initially only the clients in VLAN 2 can communicate. After enabling IP routing in the CLI, Client1 can reach the other Clients. To test the communication in the other direction from the switch to the configuration back end, we keep the setup like in the previous experiment. The CLI shows that all ports are up. When we unplug Client2, we receive a port-down event in the CLI and the port status is set accordingly.

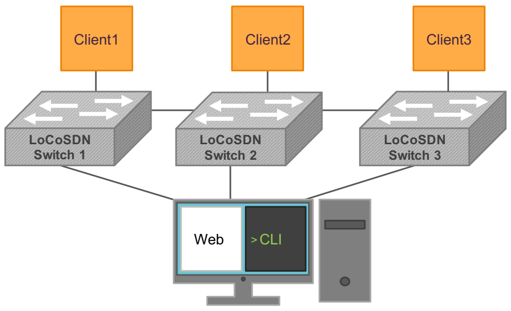


Fig. 11: The evaluation setup.

VI. Conclusion

In this paper, we presented LoCoSDN as an alternative architecture to hybrid switches. It consists of an OpenFlow-only switch and a local SDN controller running on a cheap single board computer. As with typical hybrid switches, LoCoSDN supports traditional network operation and SDN-based operation. To that end, the local SDN controller provides the basic features of a typical L3 switch which can be extended via a simple module system. In SDN mode, the SDN applications can either run within the local SDN controller or an external SDN controller with a global network view can be attached. In the latter case, the local SDN controller acts as a proxy. We evaluated LoCoSDN with the help of a prototype in a simple testbed.

References

[1] B. Gurudoss, P. Jakma, T. Teräs, G. Troxel, and Quagga developer team, "Quagga Routing Suite," http://www.nongnu.org/quagga/.

[2] Graeme McKerrell, "CLISH Documentation," http://clish.sourceforge.net/clish-0.7.3/.

[3] J. Case, R. Mundy, D. Partain, and B. Stewart, "Introduction and Applicability Statements for Internet Standard Management Framework," RFC 3410 (Proposed Standard), Internet Engineering Task Force, Dec. 2002.

[4] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011.

[5] Open Networking Foundation members, "OpenFlow Switch Specification," https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, The Open Networking Foundation, 2017.

[6] Y. Nakajima, K. Kaplita, and Lagopus developer team, "Lagopus switch," http://www.lagopus.org/.

[7] K. Rutka and FlowForwarding.org community, "LINC - OpenFlow software switch," https://github.com/FlowForwarding/LINC-Switch.

[8] NEC Corporation, "NEC ProgrammableFlow," http://www.nec.com/en/global/prod/pflow/, 2017.

[9] NoviFlow Inc., "NoviFlow NoviSwitch," https://noviflow.com/products/noviswitch/, 2017.

[10] Northbound Networks, "Zodiac FX," https://northboundnetworks.com/products/zodiac-fx, 2016.

[11] E. Verizhnikova, "SDN – Hybrid architecture," https://www.slideshare.net/verizhnikova/larch-network-hybridization-jan2013.

[12] B. Pfaff, J. Pettit, T. Koponen, E. 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 *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[13] F. Hauser, M. Schmidt, and M. Menth, "Establishing a session database for sdn using 802.1x and multiple authentication resources," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[14] Hewlett Packard Enterprise, "HP Switch Software OpenFlow v1.3 Administrator Guide nl K/KA/KB/WB 15.18," http://h20566.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=7074783&docLocale=en_US&docId=emr_na-c04777809.

[15] EdgeCore Networks Corporation, "EdgeCore OPEN NETWORKING," https://noviflow.com/products/noviswitch/, 2017.

[16] Edge Core Networks, "Open Networking Solutions for Data Center, Telecom, and Enterprise," http://www.edge-core.com/solution-inquiry.php?cls=5\&id=7.

[17] Cumulus Networks Inc. and Open Compute Project, "Open Network Install Environment (ONIE)," http://onie.org/about/, 2015.

[18] Cumulus Networks Inc., "Cumulus Linux - The world's most flexible open network operating system for bare metal switches," https://cumulusnetworks.com/products/cumulus-linux/, 2013.

[19] Facebook Inc., "Facebook Open Switching System (FBOSS)," https://github.com/facebook/fboss, 2015.

[20] A. Kuznetsov and S. Hemminger, "iproute2: Utilities for Controlling TCP/IP Networking and Traffic," http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2, 2012.

[21] Broadcom Limited, "OpenNSL," https://github.com/Broadcom-Switch/OpenNSL, 2017.

[22] ——, "OF-DPA," https://github.com/Broadcom-Switch/of-dpa, 2017.

[23] B. Cowie, C. Lorier, and J. Stringer, "valve," https://github.com/wandsdn/valve, 2016.

[24] J. Bailey and S. Stuart, "Faucet: Deploying sdn in the enterprise," *Queue*, vol. 14, no. 5, pp. 30:54–30:68, Oct. 2016.

[25] J. Bailey, "faucet," https://github.com/REANNZ/faucet, 2017.

[26] E. Gampel and DragonFlow drivers team, "dragonflow," https://launchpad.net/dragonflow.

[27] OpenStack Foundation, "Openstack," https://www.openstack.org/, 2017.

[28] opendaylight, "archived-net-virt-platform/cli," https://github.com/opendaylight/archived-net-virt-platform/tree/master/cli, 2013.

[29] OpenMUL Foundation, "OpenMUL – High Performance SDN," http://www.openmul.org/.

[30] Northbound Networks, "Zodiac FX: The world's smallest OpenFlow SDN switch," https://www.kickstarter.com/projects/northboundnetworks/zodiac-fx-the-worlds-smallest-openflow-sdn-switch, 2016.

[31] ——, "Zodiac FX Firmware," https://github.com/NorthboundNetworks/ZodiacFX, 2016.

[32] RASPBERRY PI FOUNDATION, "RASPBERRY PI 3 MODEL B," https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, 2015.

[33] M. Thompson and P. Green, "Raspbian," https://www.raspbian.org/.

[34] Ryu SDN Framework Community, "Ryu," http://osrg.github.io/ryu/.