Portable MiniLab for Hands-on Experimentation with Software Defined Networking

Portable MiniLab for Hands-on Experimentation with Software Defined Networking

Sebastian Troia[(1)], Ligia Maria Moreira Zorello[(1)], Guido Maier[(1)], Giacomo Verticale[(1)], Paolo Giaccone[(2)]

(1) Politecnico di Milano, Dipartimento di Elettronica Informazione e Bioingegneria (DEIB), Italy
{sebastian.troia, ligiamaria.moreira,guido.maier,giacomo.verticale}@polimi.it
(2) Politecnico di Torino, Dipartimento di Elettronica e Telecomunicazioni (DET), Italy
paolo.giaccone@polito.it

***Abstract*— The new paradigms brought to the networking area by softwarization, not only are revolutionizing research and industry, but start deeply impacting on teaching. While in the past most of education in basic networking relied upon studying theory and learning standard protocols, today the hands-on experience is gaining paramount importance. In this paper, after briefly explaining how SDN changed the learning experience, we present a portable, low-cost, self-contained hardware laboratory to experiment with real SDN networks based on OpenFlow switches, valuable for both teaching and research. We then show some use-case that can be investigated within possible projects developed using this testbed. Though devised mainly for SDN, we will show that this testbed can support experimentation also of traditional switching concepts, such as packet classification. Finally, we introduce a short demonstration that can be presented live at the conference.**

Keywords*— *Software Defined Networking, OpenFlow, Experimentation

I. INTRODUCTION

It is well recognized that softwarization of networks is having a major impact on modern telecommunications, not only in the domain of academic research, but also in that of the industry. This term embodies two important paradigms: Software Defined Networking (SDN) and Network Function Virtualization (NFV). This inevitably generates consequences on teaching, both in lifelong learning programs for professional staff and in university courses. Indeed, the development of networks using software sits aside or, in some cases, even replaces traditional topics of networking in courses on fundamental computer networks. As a matter of fact, widely-spread textbooks such as [1] [2] already explore this technology.

One of the key strategies in engineering education is the use of practical lessons and laboratories to apply students' scientific knowledge. Indeed, practical activities complement traditional lectures by elucidating the material taught during classes and also improving students' ability to problem solving [3]. Softwarization helps accomplishing this goal and brings the big advantage that it makes experimental and hands-on activity much easier than in the past. It introduces the possibility of direct interaction between the user and the network, being the main essence of these new trends. Consequently, the need for students and learning staff to use this technology naturally arises, which leads to great opportunities in teaching.

Several simulation and emulation tools are available nowadays, both open-source (e.g. GNS3 [4], Mininet [5]) and offered by vendors (e.g. PacketTracer [6], VIRL[7]), making hands-on and testing experience easy to achieve. However, we believe that a hardware lab, even if small-scale and low-cost, has some clear advantages compared to simulators and emulators. First, a hardware lab stimulates, coherently with well-known psychological mechanisms, young students and researchers to "build their own Internet" [8] with their hands, improving their understanding and retention. Secondly, a hardware lab has also some relevant technical advantages over an emulated environment. In fact, some kind of network measurements, typically delay, jitter and bandwidth, are hard to be performed or inaccurate in emulated or simulated networks. This is due to the influence of the computing machine (virtualization server) hosting the emulation tool, whose activity may introduce unpredictable spurious effects that make the measurements unreliable and hard to reproduce, if not meaningless in some cases.

Fig. 1 depicts the general architecture of SDN technology, divided into three layers. The higher layer consists of the Network Applications (NAs) that will be performed over the SDN controller. The middle layer, called Control Plane (CP), is based on a centralized controller that enables the underlying networking infrastructure to be abstracted. It therefore provides a programmable interface to the network. Lastly, the lowest level, called Data Plane (DP) deals with user's data, as it contains the equipment to route and transmit them to from the source to the final destination. The communication between the network applications and the CP is enabled by Northbound interfaces, such as REST API. The CP manages the DP forwarding rules and obtains information from the DP layer thanks to the Southbound interface. Several protocols can be used to implement the Southbound interface.

For the development of this testbed, we decided to adopt OpenFlow [9]. This protocol creates flow tables in switches with a set of actions to be executed according to the incoming packets. The selection of this particular protocol is due to the following reasons. Although OpenFlow is not commercially deployed everywhere (for instance, it is common within datacenters, but not in the Wide Area Networks – WANs and transport networks), it was natively developed to implement the SDN architecture. Studying OpenFlow, students can therefore have a clear perception of the SDN concept, without complications that, for other protocols, derive from back-compatibility with legacy equipment. On the other hand, OpenFlow is quite consolidated and backed by a strong

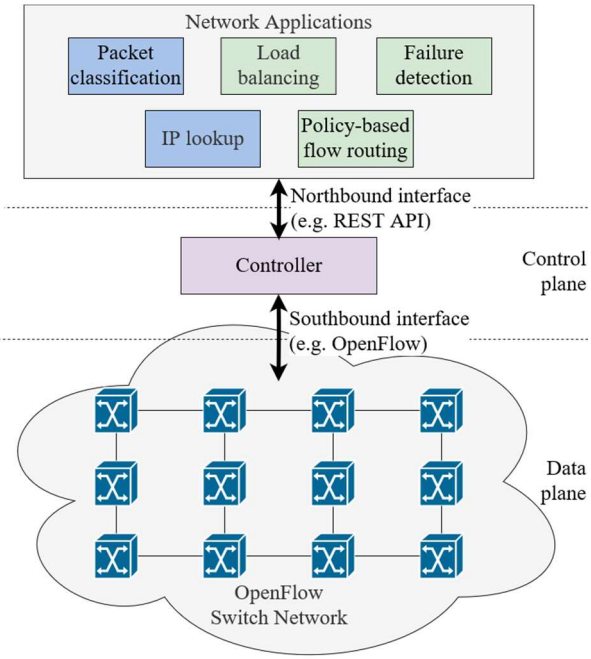


Fig. 1: General SDN architecture. It is composed by data plane, control plane and the network applications. In the picture, some examples of applications that could be implemented in the academic environment are proposed, including traditional switching (blue) and SDN (green) functions.

organization (Open Networking Forum – ONF), well documented (even in textbooks), while other more recent alternatives (e.g. P4) do not have the same stability.

As we have anticipated, the SDN paradigm have changed teaching, and it is precisely in the layered structure that we see in Fig. 1 that we can find the basis to shape the new hands-on approach. The most important feature of the SDN architecture in this perspective is having migrated the "intelligence" of the network in the network-application layer. While CP and DP are rather technology dependent, NA is completely flexible and open (assuming that the abstraction provided on the Northbound interface is powerful enough). In a teaching environment, this layer represents the "dojo" where students can better exploit their fantasy and creativeness to develop original applications to solve an ample range of networking problems. Later in this paper we will present some examples of such problems (i.e. packet classification, IP lookup, load balancing, failure detection and policy-based flow routing). So, the conception of our lab is to provide students with a CP and a DP ready to be used, leaving them to focus on the development of their network applications.

Another strength of SDN is the availability of many CP implementations based on open-source projects: as detailed later on, we relied completely on open-source to implement the CP in our lab. What is instead more peculiar is the implementation of the DP, which we decided to build with hardware. In this paper we focus on the small hardware laboratory, namely MiniLab, that, despite its simplicity and low cost, allows us to reproduce a real SDN network. It was the result of a cooperative work between research groups of Politecnico di Milano (Italy) and Politecnico di Torino (Italy). MiniLab enables the student to learn and practice two key operations for real networks: 1) how to configure and setup various network topologies of switches with their control plane (e.g. using various SDN controllers); 2) how to implement and test complex network functions and assess their actual performance. Based on the experiences of regular courses on network engineering in both universities, MiniLab was considered to be useful to students for hands-on teaching activities. Also, it brought benefits to researchers that are able to test algorithms and control-plane solutions. In addition, thanks to the reduced size of the lab can be easily portable to other places, facilitating its use both in lectures and dedicated laboratory practices.

The remaining of the paper is organized as follows: Sec. II describes technical aspects of the testbed, Sec. III explains possible projects that could be developed on this testbed, Sec. IV presents two demonstrations that will be shown live during the paper presentation as examples of possible experiments, and Sec. V concludes the paper.

II. Lab Description

The network environment was developed by the BONSAI Research Group of Politecnico di Milano with collaboration of Politecnico di Torino. The testbed comprises 12 Zodiac FX Northbound switches [10] based on the OpenFlow protocol [9] and 6 hosts that can act as servers and clients for network services. The controller accesses the forwarding plane of the switches by the OpenFlow communication protocol (Southbound interface). In order to forward traffic from one port to another of a switch, the controller installs a flow rule in the flow table of the switch.

Any SDN controller supporting OpenFlow on the Southbound can be used in this laboratory. Among several available open-source controllers, we selected the Ryu SDN controller [11]. Ryu is a component-based SDN framework application developed in Python. It is particularly suitable for educational purposes thanks to its simplicity, flexibility and reliability. It provides a well-defined Application Programming Interface (API) that allows users to easily develop their network applications on the top of Ryu, using the Python programming language, which is easier to learn and use than other languages. Other SDN controllers more sophisticated than Ryu can be used for more advanced projects. In particular, the Java-based ONOS [12] and OpenDaylight [13] controllers have already been tested in the MiniLab and used in research projects.

In order to experiment with the SDN network testbed, we adopted Raspberry PI 3 model B+ computers to act as network hosts (see Fig. 2). Each Raspberry is a full-fledge computing system, but it is small in size and consumes a limited amount of energy. Finally, this device supports generic Linux Debian-based applications, allowing a wide spectrum of network services and applications to be executed.

In Fig. 2 we show the physical outline of MiniLab. Just for sake of illustration, in the picture the OpenFlow nodes are connected on the data plane in such a way as to form three adjacent rings, but other physical topologies can be easily implemented with a proper cabling. The control port of each node is connected to the controller via a cable (yellow lines in Fig. 2) and through two standard Ethernet switches. In this way, we can install the SDN controller in an external laptop or PC.

This solution gives students the opportunity to configure and modify the controller, and develop their network applications. They can also work at home, by testing their software offline (emulating the testbed e.g. throughout

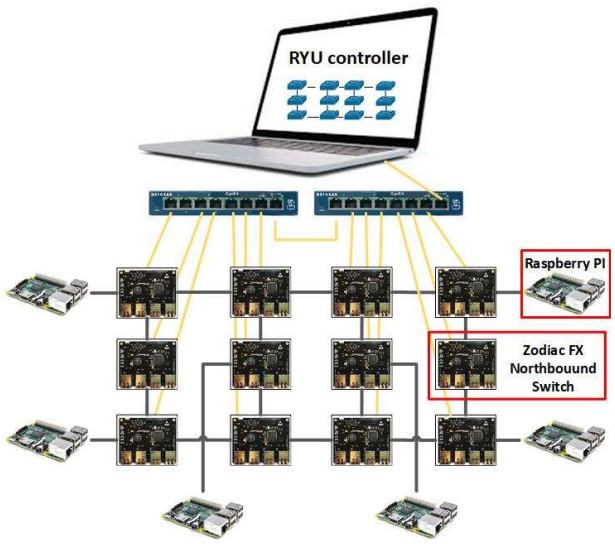
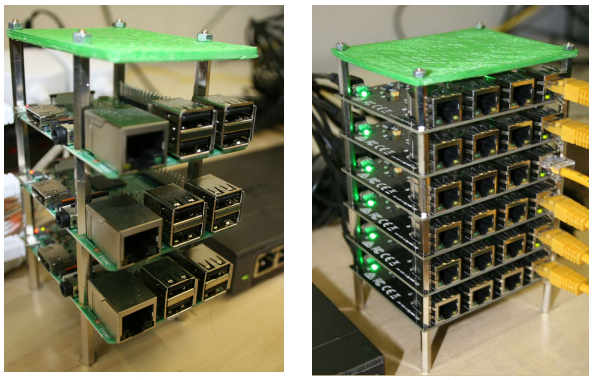


Fig. 2: Schema of MiniLab. Gray lines are the Ethernet cables for the data plane links. Yellow lines are the Ethernet cables used for the control plane. The laptop runs the SDN controller and it is connected to all the OpenFlow nodes through Ethernet cables and two interconnected standard Ethernet switches. Each host is connected directly to one OpenFlow switch.

Mininet), and then come to the lab with their software ready to be installed. Since every Raspberry PI has a built-in Wi-Fi dongle, we have setup an ad-hoc Wi-Fi management network, inside which we can configure the hosts and the services provided by the hosts.

Finally, the configuration of the switches operates either via GUI, or via Command Line Interface (CLI) through a serial communication link (COM port). At the moment, the configuration using protocols such as OF-Config [14] is not supported, but it will be added in future developments. In order to promote the compactness and portability of the testbed, we have divided the MiniLab into 4 mini-rack modules. Each mini-rack contains either 3 hosts (Fig. 3a) or 3 to 6 switches (Fig. 3b). The racks can be used in different manners to operate smaller or larger topologies, according to the users' needs.

Table I summarizes the hardware specifications of MiniLab, specifying the processor, memory, connectivity, features and input power used.



(a) (b)

Fig. 3: Photo of MiniLab. The image on the left (a) depicts a mini-rack module with three Raspberry PI hosts, while the one on the right (b) shows a mini-rack module mounting six Zodiac switches.

III. Use Cases

As explained in Sec. I, the use of SDN technology as a teaching tool can bring several benefits to learning. First of all, such a laboratory enables visualizing in a real scenario the application of algorithms that were studied during lectures in a simplified manner. Furthermore, students are able to solve problems related to networking by studying, planning and implementing different algorithms and network applications. A number of projects can be exploited in such a laboratory. For instance, students can virtualize traditional switching functions, such as: IP-address lookup and packet classification. On the other hand, thanks to this environment, they are also able to implement network applications that performs typical SDN network functions, such as: MPLS routing, load balancing, node and link failure detection, policy-based flow routing, etc. In both cases, students have to implement in practice and with "their own hands" the algorithms that they learn in the courses, directly experiencing in a comparative way concepts such as performance and complexity.

In this section, we describe some use cases divided into two scenarios. The first one comprises traditional networking function such as packet classification. The second presents projects based on SDN, considering failure detection and recovery mechanisms, and dynamic flow routing.

TABLE I. HARDWARE SPECIFICATIONS OF MINILAB

Characteristics	**Raspberry PI**	**Zodiac FX Switch**
Processor	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz	Amtel SAM4E8CA. Microchip KSZ8795CLX Managed Ethernet Switch
Memory	1GB SDRAM	128KB SDRAM
Connectivity	2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0 max throughput 300Mbps 4 x USB 2.0 ports	4 x 10/100Mbs Ethernet Ports USB Serial (COM) and Web-Based Configuration
Features	*Size*: 87 x 58,5mm *Weight*: 49,7grams SD card support: Micro SD format for loading operating system and data storage	*Size*: 100 x 80mm *Weight*: 115 grams Compatible with OpenFlow 1.0 & 1.3
Input Power	5V/2.5A DC via micro USB connector	5V/300 mA DC via micro USB connector

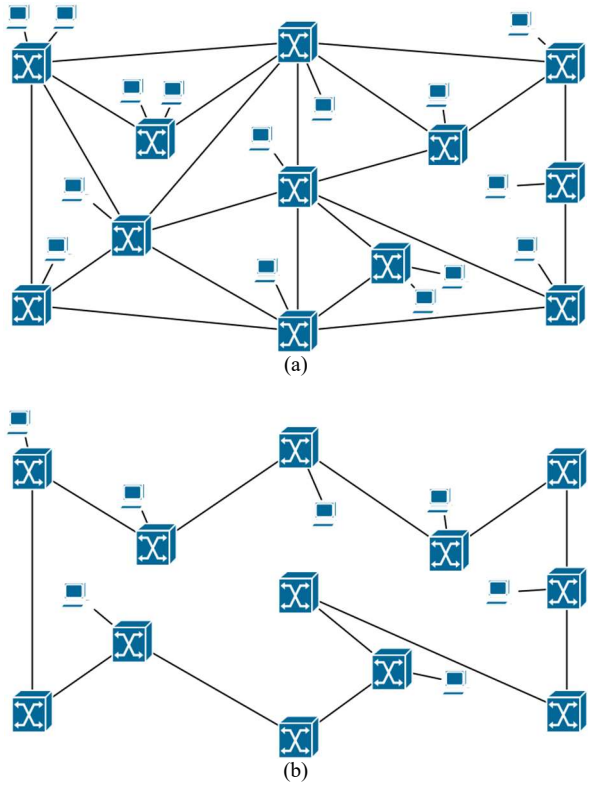


Fig. 4: Example of topology for packet classification project to be implemented in Mininet (a), and a simplified version for MiniLab (b), which only supports ring topologies and maximum of six hosts in its current status.

A. *Traditional switching functions: packet classification with geometric algorithms*

The objective of this project is that students learn how to implement, test and evaluate packet classification algorithms, normally performed by switches, routers and other network equipment to identify flows of packets by recognizing regular expressions in the headers. This function is implemented in the firmware or the operating system of the switches (incidentally, Zodiac switches, being OpenFlow-based, themselves internally perform packet classification). In order to allow students to develop their own classification function, we have to exploit a gimmick. OpenFlow allows us to instruct the switches to send incoming packets to the controller, instead of processing them directly. So, in this project the switches send to the controller a large number of packets. The controller classifies the packets and return them to the switches. In this way, students can develop packet classification as a network application on the top of the controller.

Several algorithms have been proposed in literature for packet classification: an ample selection is reported, for instance, in [15]. In this paper, we take as example those based on geometric classification, and, in particular, two implementations known as cross-product [16] and bitmap intersection [17]. In this project, students implement the two algorithms in the NA layer over the Ryu controller, and then test the algorithms in a network topology to understand their differences in terms of performance (lookup computation time and memory usage).

In the first step, the project is done in emulated environments such as Mininet. The development of the algorithms and their testing in Mininet can be done on their own. Mininet enables creating large topologies such as the one illustrated by Fig. 4a. By instantiating several end-to-end flows in this virtual testbed, they can generate a large number of packets with many different header attributes, which is good to test the algorithms with enough statistical confidence. Classifier tables can also be easily scaled to include a variable number of rules. At the end of the first step, the collected data allows us to experimentally compare the complexity of the algorithms.

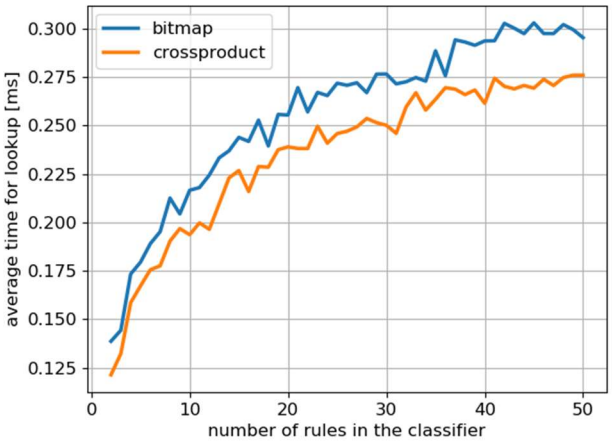


Fig. 5: Scalability comparison of average packet-classification time in miliseconds of bitmap and crossproduct geometric algorithms according to the number of rules used in the classifier.

In the second step, the physical testbed can be used as a small-scale environment to verify how the developed network apps work on a physical DP. Since the current version of MiniLab has a limited number of OpenFlow switches, ports and hosts, it does not allow the same statistic confidence as in Mininet. , The topology must be modified, e.g. as represented in Fig. 4b, to cope with the testbed limitations (only ring topologies can be implemented). Nevertheless, it complements students learning thanks to the implementation and evaluation of the classification algorithms in a real environment, contributing also to the easiness of the results visualization and analysis.

Fig. 5 shows one of the results expected after the development of this project. The average time that bitmap and cross-product algorithms take to calculate the look-up is very similar, although cross-product presents better scalability. Results are in good accordance with complexity analysis. Classification time-time complexity is $O\ (d \cdot t_{RL})$ for the crossproduct and $O\ (d \cdot t_{RL} + d \cdot N / w)$ for the bitmap, where N is the number of rules, d is the number of classification fields, w is the number of bits used in the bitmap and t_{RL} is the time complexity of finding a range in one dimension. In this case, $d = 2$ and is $O\ (log_2\ 2N)$, since a binary-tree search was used to perform range lookup in one dimension. It is important to notice that, since the algorithms are embedded in the controller and not in the switches, students obtain the similar curves when implementing them in a real testbed or in an emulated scenario.

B. *SDN Projects*

The projects explained in this section aim at studying SDN applications considering two types of operations. The first one illustrates a scenario for verifying failure detection and

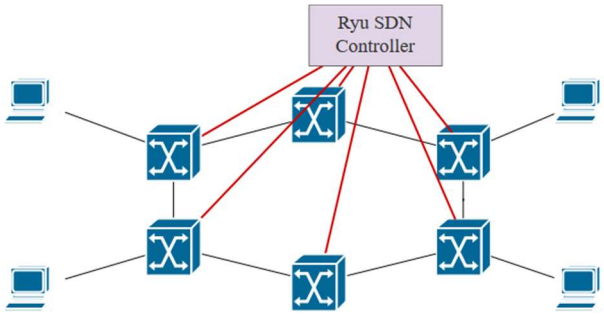


Fig. 6: Ring topology with six switches and four hosts managed by a Ryu SDN Controller.

recovery operation. The second one consists of testing a flow routing algorithm based on dynamic weights associated to the carried traffic. They consider the network depicted in Fig. 6 as an example, which illustrates a ring topology containing six OpenFlow switches connected to four hosts, and managed by a Ryu SDN Controller.

As previously stated, students can develop a series of tests to understand multiple functionalities and operation modes of SDN in a physical testbed such as MiniLab. Moreover, in order to provide a complementary activity and optimize the use of the lab amongst the multiple groups, the experiment can first be implemented in Mininet. Consequently, they are able to build an emulated topology and test the controller and application codes before preparing the real scenario. After having tested the controller in an emulated environment, students can recreate the same topology by connecting the available switches and initializing the controller and the application. This set of activities helps students obtaining an extensive comprehension on the development and implementation of SDN solutions.

1) Failure detection and recovery

The overall objective of projects on failure detection and recovery is to design a centralized system that can react to node and link outages in a topology in such a way that it is able to restore the connection automatically. To this end, students must deploy failure detection and recovery mechanisms in the controller, and generate a link or node failure to evaluate the network reaction.

The workflow for the development of this project is the following. In the first step, students implement a ring topology, such as the one shown in Fig. 6. Then, they configure Ryu SDN Controller, including the design of functions to fill-in flow and group tables in switches in order to enable connection recovery in case of failure. After that, all hosts should be able to communicate among themselves. Next, a link (or node) fault is caused by physically removing the connection between two nodes in order to test the network operation in the event of a malfunctioning. In Mininet, this fault generation is performed through a script, while in the physical testbed, one of the cables is manually disconnected from a switch.

The optimal operation mode is illustrated by Fig. 7. Initially, hosts H1 and H2 communicate with each other using a single link (see Fig. 7a). In the case in which this link fails, the system is able to recover the communication among the hosts by the selection of another flow routing path, as depicted in Fig. 7b.

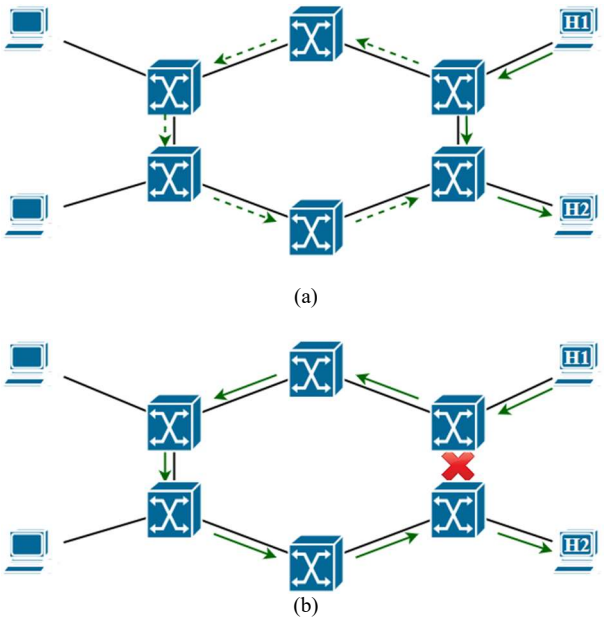


Fig. 7: Failure detection and recovery mechanism in a ring topology. The image on the top shows its normal operation, in which a working (solid) and a backup (dashed) path are calculated for host 1 (H1) to transmit data to host 2 (H2). Figure (b) illustrates a failure in the link previously used to communicate, and the SDN network is able to automatically switch to the backup path.

2) SDN with dynamic routing

The main goal of this use case is to dynamically calculate SDN flows according to the traffic information of each connection to enable load balancing. For this, students are requested to deploy an SDN topology in which Djikstra algorithm is used to dynamically compute the optimal path between nodes of a weighted network.

The first step of this project is to implement an exemplary ring topology, such as Fig. 6, applying weights for each link as proportional to their current available bandwidth. The students then configure Ryu SDN Controller in such a way that it is able to recalculate the optimal path using Djikstra algorithm upon packet arrivals. Each flow table is therefore immediately updated according to the selected flow routing. Next, they can evaluate the correct functioning of the system by opening connections between several hosts and, hence, evaluating the different flow routes assigned to these connection requests.

Fig. 8 exemplifies the expected outcome of this implementation. At first, H1 requests a connection to H2, the shortest path is calculated, and information can be exchanged

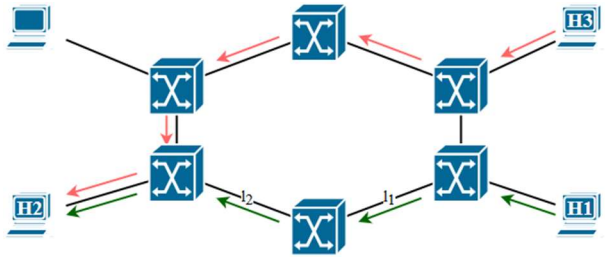


Fig. 8: Dynamic routing mechanism. After the green flow path is assigned between H1 and H2, the controller updates the weights of links l_1 and l_2, and new flow path is assigned avoiding these links via the pink path.

via the green path. The controller then updates the link weights considering the current network usage, i.e. the available bandwidth of l_1 and l_2 decreased. Next, when a new demand is requested among H3 and H2, the controller takes into account current weights to compute the new path, shown in pink.

IV. Demo Experiments

We developed two demonstrations to exemplify the usage of MiniLab. They involve a self-healing ring and the measurement-based flow allocation.

1) *Self-healing ring*

Fig. 9a depicts the testbed setup for the self-healing ring experiment. We connected all OpenFlow switches together in a ring topology using the white cables. We also connected three Raspberry Pi hosts to three switches to monitor whether packets reach their destination. The controller was attached to the switches by the yellow cables.

In this experiment, Ryu controller first retrieves the physical topology and preconfigures the network to route the traffic clockwise from a source to a destination host. Ryu provides an interface (Ryu Topology Viewer) that shows the topology discovered by the controller (see Fig. 9b) and the flow tables installed in each switch. In order to evaluate that the packets are being correctly transmitted, we use the ping command from one host to another. The switch where the destination host is located forwards the packet from the ring to the destination, and we therefore observe that it is indeed reachable.

Then, we simulate a single link failure by physically disconnecting a random link, as shown in Fig. 9c. The controller can no longer retrieve the link between two of the hosts, and updates all flow tables. Fig. 9d illustrates the new topology from Ryu Topology Viewer without the link that was manually detached. By analyzing the ping execution, we observe that the destination continued to be reached. Hence, the SDN controller is able to automatically restore the connection between the two hosts after a link failure.

This experiment enables students to test and to compare different restoration strategies, such as reactive and proactive ones. In the reactive strategy, when the switches report the link failure, the controller replaces existing forwarding rules and configures the new path. In the proactive strategy, the controller installs the protection path in advance and configures the interfaces to send packets counter-clockwise in case of a local failure.

2) *Measurement-based flow allocation*

In the second experiment, we demonstrate a measurement-based flow allocation, in which the physical network is configured as a partial mesh. The controller performs topology discovery, and periodically collects link usage measurements from the switches. This data is used to build a graph of the network with link weights that dynamically change over time according to the real-time link usage. When the switch reports a new incoming TCP connection to the controller, the controller uses this network graph to find the minimum congested path to the destination host. Then, it installs the desired path on the switches according to the flow rule that routes all the connection packets through the chosen path. This experiment enables showing the effectiveness of self-adapting routing policies implemented in the SDN controller.

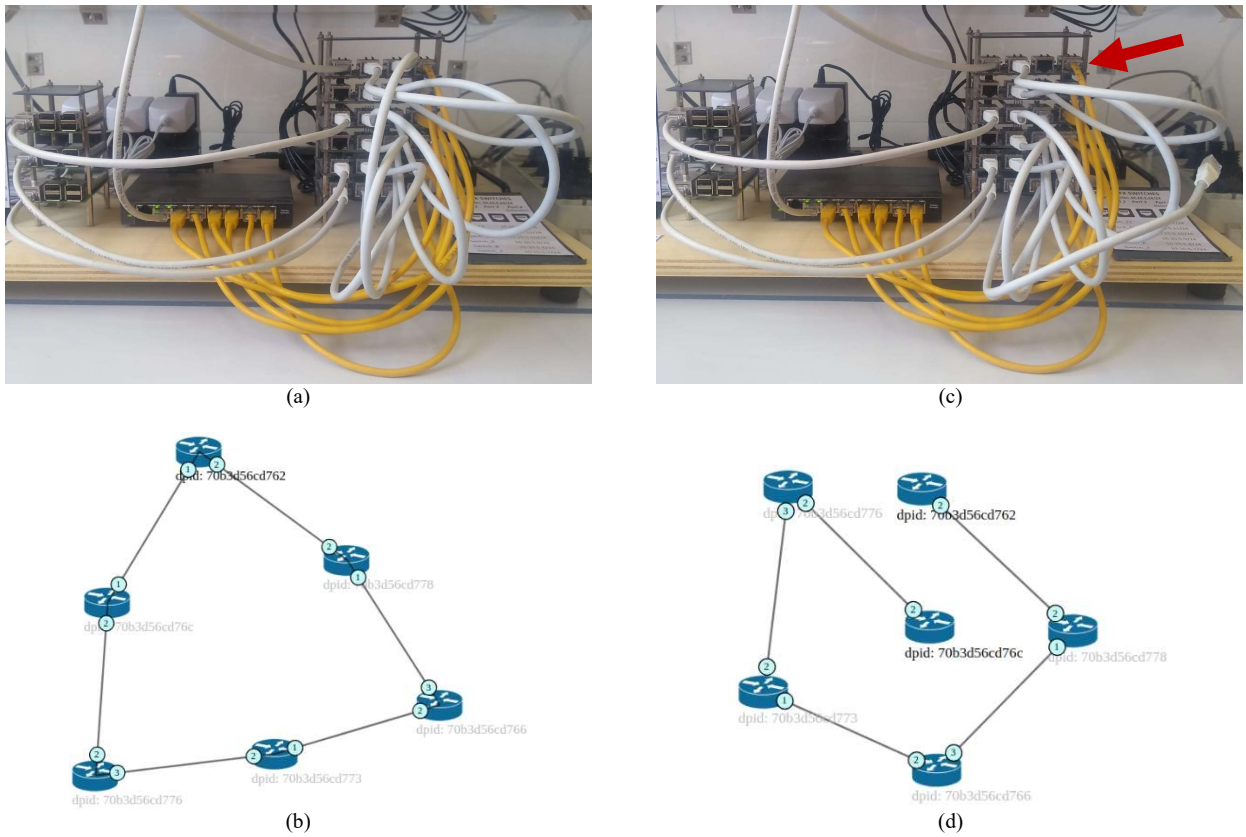


Fig. 9: Self-healing ring demonstration. Figure (a) presents the testbed setup with OpenFlow switched connected among themselves and to three Raspberry Pi hosts to create a ring topology. Figure (b) shows the topology retrieved by the SDN controller in Ryu Topology Viewer. Figure (c) illustrates the link failure with the disconnection of a cable from a switch, which is depicted by Ryu Topology Viewer as Figure (d).

V. Conclusions

We presented MiniLab, a self-contained and compact laboratory tailored to experiment with real SDN networks. MiniLab connects one SDN controller to 12 OpenFlow switches and controls the traffic between 6 hosts. All the involved hardware devices (Zodiac FX Northbound switches and Raspberry PI computers) are low cost, but fully programmable. Despite its simplicity, MiniLab is flexible enough to perform a large number of experiments, useful not only for teaching networking, but also for research in many SDN and NFV scenarios. We described some use cases related both to traditional switching and to the SDN scenario to explain possible usages of this testbed. Thus, we expect it to be an effective tool to learn basic and advanced networking techniques, with a broad applicability, from optical to wireless networks.

As future work, we expect that the adopted modular approach will be scaled to larger networks, involving also different communication technologies and mimicking key architectures and control schemes of future 5G networks.

Acknowledgment

The work leading to these results has been supported by the European Community under grant agreement no. 761727 Metro-Haul project and by Politecnico di Milano funds for innovative teaching. The authors thank the students of the "Software Defined Networking" and "Switching and Routing" classes.

References

[1] J. F. Kurose, K. W. Ross, Computer Networking: A Top-Down Approach, 7th ed, Pearson, 2017.

[2] W. Stallings, Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud, 1st ed, Addison-Wesley Professionalm, 2015.

[3] A. S. Vaughan, P. L. Lewin, A. M. Macdonald, "Virtual experiments: Providing students with a unique online laboratory experience," in European Workshop on Microelectronics Education (EWME), Southampton, UK, 2016, pp. 1–4, 2016.

[4] GNS3, "The Software that Empowers Network Professionals," Internet: https://www.gns3.com/ [Apr. 11, 2019].

[5] Mininet Team, "Mininet," Internet: http://mininet.org/ [Apr. 11, 2019].

[6] Cisco, "Cisco Packet Tracer," Internet: https://www.netacad.com/courses/packet-tracer [Apr. 11, 2019].

[7] Cisco, "Virtual Internet Routing Lab," Internet: http://virl.cisco.com/ [Apr. 11, 2019].

[8] L. Yan, N. McKeown, "Learning Networking by Reproducing Research Results," ACM SIGCOMM Computer Communication Review, vol. 47, n. 2, pp. 19–26, 2017.

[9] N. McKeown et al., "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review, vol. 38, n. 2, pp. 69–74, 2008.

[10] Northbound Networks, "ZODIAC FX," Internet: https://northboundnetworks.com/products/zodiac-fx [Mar. 11, 2019].

[11] RUY SDN Framework, "Component-Based Software Defined Networking Framework: Build SDN Agilely," Internet: https://osrg.github.io/ryu/ [Mar. 11, 2019].

[12] P. Berde et al., "ONOS: Towards an Open, Distributed SDN OS," in: ACM HotSDN, Chicago, USA, 2014, pp. 1–6.

[13] J. Medved, A. Tkacik, R. Varga, K. Gray, "Opendaylight: Towards a Model-Driven SDN Controller Architecture," in IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014, pp 1–6.

[14] R. Krejci, T. Cejka, M. Vasko, S. Natarajan, "OF-CONFIG interface to Open vSwitch," Internet: https://github.com/openvswitch/of-config Sept. 9 2015 [Mar. 11, 2019].

[15] H. Jonathan Chao and Bin Liu, High Performance Switches and Routers, 2007 John Wiley & Sons, Inc.

[16] V. Srinivasan, G. Varghese, S. Suri, M. Waldvagel, "Fast and scalable layer four switching," in: ACM SIGCOMM, Vancouver, Canada, 1998, pp 191–202.

[17] T. V. Lakshman, D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in ACM SIGCOMM,Vancouver, Canada, 1998, pp. 203–214.