SDN Testbed for Undergraduate Education

Weerawardhana J.L.M.N.*, Chandimal N.J.A.W.†, Bandaranayake A.‡
Department of Computer Engineering
Faculty of Engineering
University of Peradeniya
Peradeniya, Kandy
*jlmadushan@gmail.com,†waruna.rdp@gmail.com,‡asithab@pdn.ac.lk

Abstract—**Software defined networking (SDN) is a new paradigm of networking that can change the way of networking. It is a modern approach to designing, building, and managing networks that separates the networks control (brains) and forwarding (muscle) planes to better optimize each. With a centralized controller and very simple data forwarding units, SDN offers very easy way of defining policies of the network and flexible ways of controlling traffic flow.**

Objective of this project is to build a SDN Testbed for educational and research purposes using commodity embedded hardware like Raspberry Pi. The user will be able to perform normal network functions without understanding SDN concepts and understand SDN concepts if needed using this SDN testbed. And the ultimate target here is to make an educational tool which offers a visualization of traffic flows and several advanced network functionalities like network slicing, dynamic resource allocation and network function virtualization.

I. Introduction

Software Defined Networking (SDN) tries to address several problems in traditional IP networking, specially the speed of evolution of network technologies by providing the separation needed between network's control logic (control plane) and actual data forwarding (data plane). By doing this, all the complex network devices like switches, routers or firewalls can be replaced by a simple and inexpensive forwarding device since low level network devices does not have to take any of the traffic control decisions. As an added advantage, all the control plane actions can be performed by one centralized controller, essentially simplifying network management functions like flow controlling or rule based policy enforcement.

Provided that SDN can bring down the cost and complexity of building a new network, converting a traditional network to an SDN is not that easy. Infrastructure costs on OpenFlow [1] switches and controller hardware/software can overwhelm the scope of a simple research project. Specially if the project is done in a undergraduate level.

Software Defined Networking testbed for undergraduate education aims to create a simple, self contained and portable network testbed. The basic motivations behind this was to create a sophisticated educational tool to teach the core SDN concepts and use as a low cost SDN research platform without converting the existing infrastructure to an SDN.

The background chapter of the paper talks about used technologies, software and related work. Proposed work chapter is about work done trough out the project and Conclusion chapter delivers the closing remarks and future plans.

II. Background

There has been a number of attempts to create a low cost SDN research platforms in the past based on both network virtualization and low cost hardware. Other than that there are SDN testbeds for large scale research like OF@TEIN [2] or OFELIA [3] built on top of inter continent education and research networks.

A. Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking [4]

An attempt to create a cost effective SDN testbed using low cost hardware. The project utilize Raspberry Pi devices with Open vSwitch configured as the OpenFlow switch and Floodlight as the OpenFlow controller.

B. Mininet [5]

A virtual network based approach to building dynamic SDN testbed. Employs Linux lightweight process containers and cgroups in order to create lightweight virtual hosts and creates SDN capable virtual networks using Open vSwitch. A wide verity of tools such as OpenFlow controllers, Packer inspectors, performance monitoring tools are pre packaged with mininet. Highly documented and extensible.

First one of the above mentioned projects (subsection II-A) uses a smiler approach we are using. But one of the main differences is that it doesn't provide any of the documentation needed to reproduce their work by an independent researcher or undergraduate student. This makes it difficult to employ their work in a ad-hoc manner. Additionally the education aspect of the our work is not covered in this work at all. It does not provide any methodology or documentation needed to teach the basics of SDN to a beginner.

Mininet (subsection II-B) on the other hand have a extensive set of documentation and Wiki entries. Mininet provides a simple methods of building, altering and controlling SDNs and experiment on them. But it doesn't provide realistic environments or realistic statistics.

In our work we provide a methodology to build a low cost SDN testbed and the necessary documentation. And the methodology and documentation to introduce SDN concepts to a beginner using this SDN testbed.

III. Solution Overview

There are several things to be considered when building a SDN based on real low cost hardware. First one being the setup and configuration difficulties occurs with physical devices. Unlike virtual network approaches, before beginning to build a software defined network, physical networks require a infrastructure setup for reasons like the OpenFlow protocol's dependency on TCP connections to transfer OpenFlow control packets.

A. *OpenFlow*

The standard our SDN testbed built upon. OpenFlow standard implements the idea of OpenFlow switches, which are just dumb forwarding devices. And OpenFlow controllers, which are the separated control plane, most of the time living in a separate server, injecting the necessary rules to forward the data traffic and serving the SDN application requests. Fig. 1 shows the basic architecture of a OpenFlow SDN.

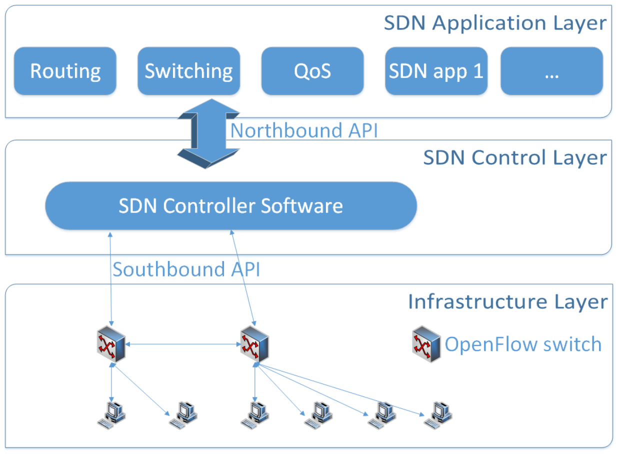

Fig. 1. OpenFlow SDN architecture

OpenFlow SDNs use OpenFlow protocol to communicate control data between the controller and the OpenFlow switch. This protocol works by altering the flow table of the OpenFlow switch exposed via OpenFlow southbound API. Fig. 2 shows a OpenFlow flow table header.

When a packet is reached an OpenFlow switch, Switch tries to match the header values from the packet to one of the rules defined in the flow table. If a match is found, the action corresponding to it carried out. If the packet is a mismatch, It's wrapped in a OpenFlow header and forwarded to the controller. Controller inspects the packet and broadcasts new rules based on the policies defined by SDN applications.

Switch port	MAC src	MAC dst	Eth type	VLAN ID	IP src	IP dst	IP port	TCP sport	TCP dport	Action

Fig. 2. OpenFlow flow table entry header

B. *Physical network*

Fig. 3 is the proposed network topology for the physical network. The routing elements in the topology are just represented as such to denote intra domain routing capabilities. The only network element used in OpenFlow software defined networking is OpenFlow switches. So the routers are actually OpenFlow switches carrying out routing tasks.

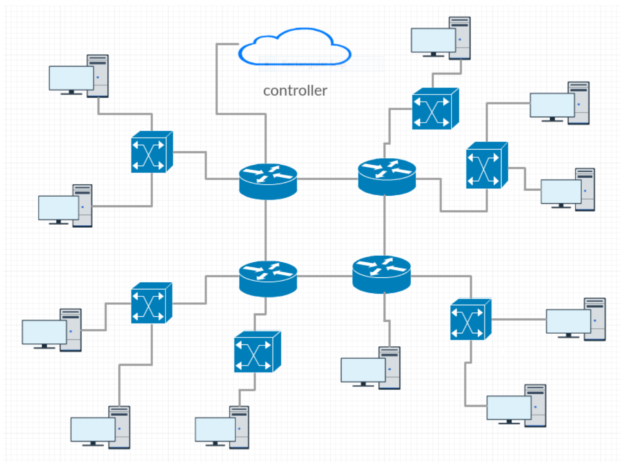

Fig. 3. Proposed network topology

C. *Raspberry Pi*

We decided to build our SDN switches on Raspberry Pi devices which has the necessary specs for our tasks. Raspberry Pi is a credit card size computer commonly used for embedded computation tasks. The perticuler version of the device we are using is Raspberry Pi v2B which has a 900MHz quad-core ARM Cortex-A7 CPU and 1GB of RAM. It has one 100mbit ethernet port and 4 USB ports which we can use as network interfaces with USB to ethernet converters.

D. *SDN switch*

Every OpenFlow switch in the network is a Open vSwitch instance running on a dedicated Raspberry Pi.

Raspberry Pi devices only have one ethernet port, but according to the network topology, some switches on the network needed to have four network interfaces. This problem was solved using USB to ethernet converters. Fig 4 shows the model of the SDN switch used.

Open vSwitch instances running on Raspberry Pi devices does not inherently have the control of the network interfaces available on the device. A special bridge device needed to be created and each network interface needed to be controlled by the Open vSwitch instances should be added to this bridge device. The bridge device has a unique ID called "datapath ID" in order for the controller to recognize them separately.

Controller for each bridge device can be specified separately. In this case bridges communicate with the controller over standard OpenFlow port (tcp:6633). Due to a kernel incomparability problem, stable version of the Open vSwitch software could not be used with the device. So, the development version of the software used instead.

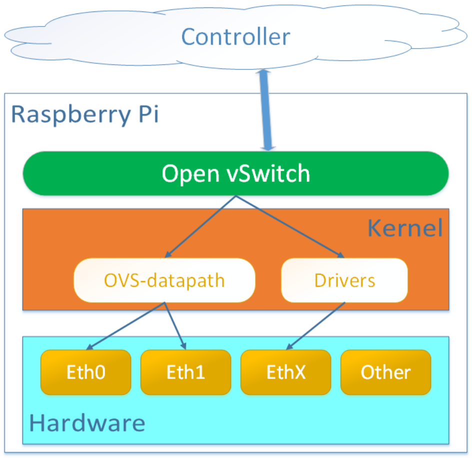

Fig. 4. SDN Switch System Model

E. *The SDN controller*

OpenDaylight controller framework version Helium karaf distribution was configured with the following OpenDaylight modules. Fig 5 shows a model of the SDN controller used.

- odl-l2switch-switch – layer two switching
- odl-l2switch-ui – layer two switch information and configuration in DLUX web ui
- odl-restconf – northbound RESTful APIs for the controller
- odl-dlux-core – DLUX web ui for network visualization and configuration
- odl-mdasal-apidocs – API documentation for model-driven service abstraction layer (MD-SAL)

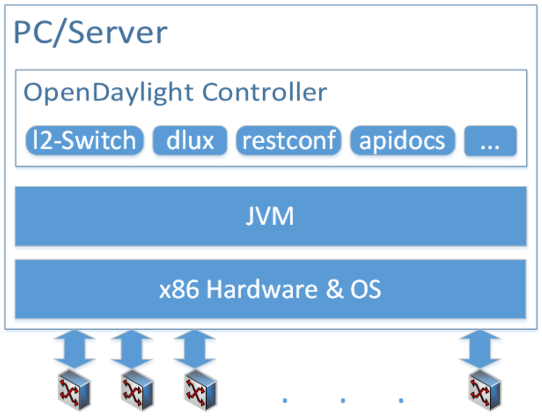

Fig. 5. Controller System Model

F. *SDN functionality*

Upon the successful establishment of tcp connection to controller from Open vSwitch, since old-l2switch-switch is installed on the Opendaylight controller, Open vSwitch starts acting as a normal layer two learning switch.

Every Raspberry Pi device in the network was connected to the controller directly using independent ethernet connections. This approach provided easier debugging capabilities but in a practical deployment this is not viable because it requires a separate non SDN layer two switch. As a solution “static routing” functionality was tested to connect the controller to the network only at one point but lated abandoned because of stability problems.

After setting up the network, primary SDN characteristics like dynamic forwarding rules and host tracking capabilities were demonstrated and the procedures were documented so anyone can regenerate the results for the “educational tool” aspect of the projects.

IV. Conclusion

In the work covered in this paper, we built a low cost SDN testbed can be used for research and educational purposes using Raspberry Pi as a low cost hardware platform. We employed Open vSwitch as the SDN switch implementation and OpenDaylight as the SDN controller framework. as opposed to the previous projects done in this area, we provide complete methodology and documentation for building a similar SDN testbed for interested third parties. Additionally, we also provide the methodology and documentation to introduce basic SDN concept to a beginner using this testbed by demonstration.

The tested performance statistics were much lower comparing to a network built on a Hardware OpenFlow switches like NetFPGA because of the speed limitations of USB to Ethernet converters and Raspberry Pi processor. But for non performance critical SDN research and educational purposes statistics were in a acceptable range.

In the future, The SDN testbed will be extended in to a full featured small-scale SDN offering additional features like traffic visualization, network slicing, dynamic resource allocation or network function virtualization.

Acknowledgment

This research project was entirely funded by the Association of Computer Engineering Students (ACES), University of Peradeniya.

References

[1] 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.

[2] J. Kim, B. Cha, J. Kim, N. L. Kim, G. Noh, Y. Jang, H. G. An, H. Park, J. Hong, D. Jang *et al.*, “Of@ tein: An openflow-enabled sdn testbed over international smartx rack sites,” *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, pp. 17–22, 2013.

[3] A. Köpsel and H. Woesner, “Ofelia: Pan-european test facility for openflow experimentation,” in *Proceedings of the 4th European Conference on Towards a Service-based Internet*, ser. ServiceWave’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 311–312. [Online]. Available: http://dl.acm.org/citation.cfm?id=2050869.2050905

[4] H. Kim, J. Kim, and Y.-B. Ko, “Developing a cost-effective openflow testbed for small-scale software defined networking,” in *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, Feb 2014, pp. 758–761.

[5] R. Morling and G. Cain, “Mininet: a packet-switching minicomputer network for real-time instrumentation,” in *Proc. AIM International Meeting on Minicomputers and Data Communications, Liège, Belgium (January 1975)*, 1975.