ACTA UNIVERSITATIS AGRICULTURAE ET SILVICULTURAE MENDELIANAE BRUNENSIS

Volume 63 228 Number 6, 2015

http://dx.doi.org/10.11118/actaun201563062093

FIRST EDUCATIONAL STEPS IN SDN APPLICATION DEVELOPMENT

Martin Pokorný[1], Petr Zach[1]

[1] Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic

Abstract

POKORNÝ MARTIN, ZACH PETR. 2015. First Educational Steps in SDN Application Development. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis,* 63(6): 2093–2099.

Software Defined Networking (SDN) is a new approach in design, implementation and maintenance of computer networks. With SDN, it is possible to dramatically reduce business costs because the whole network can be maintained in a centralized fashion which leads to a simplified and effective network management. This paper is concerned with SDN from educational point of view. The paper's primary goal is to present beginning of a learning path to start experimenting with SDN programming and traffic path definition based on Hewlett-Packard (HP) SDN solution. Useful sources of documentation were selected and two experiments were performed: firstly an already existing HP SDN application was extended in the area of network visualization, secondly and experiment of traffic path definition with real network switches was performed.

Keywords: computer network, software defined networking, openflow, Hewlett-Packard

INTRODUCTION

Computer network architectures tend to centralize their control. This approach was adopted into wireless networks by means of wireless controllers a few years ago, nowadays, the same trend is coming into wired networks by means of Software Defined Networking (SDN). Even though the traditional hierarchical network design has proved its usability and effectiveness, the idea of a central point of network management and programmability is tempting, and it is able to reduce network deployment and maintenance costs in the long-term perspective.

Software Defined Networking represents a recently announced and dramatic change in the way how to design, implement and maintain computer networks, and the networking community is awaiting more information how to incorporate SDN into daily networking routine. The goal of this paper is to present our first SDN applications based on Hewlett-Packard SDN application prototypes, guidelines, controller and API. The overall objective of our paper is mainly educational – to facilitate SDN learning path e.g. for our students.

MATERIALS AND METHODS

Second chapter is organized as follows: firstly the basic theory of SDN is presented, after that comes technological description of the Hewlett-Packard SDN solution from the programming point of view, including references to several useful guides and tools, in the end, our two test cases are introduced.

Software Defined Networking

Software Defined Networking (SDN) is promoted by the Open Networking Foundation (ONF); among its members there can be found e.g. Cisco Systems, Facebook, Google, HP, Intel, Juniper Networks, Microsoft, Oracle, Samsung, VMware, and many others.

According to the Open Networking Foundation (2012), there are several limitations of current network technologies, e.g. network complexity caused by device-level management, static nature of today's networks that cannot adapt to changing traffic and user demands dynamically, difficulties to apply consistent security and QoS policies, scalability problems, and vendor dependence. The ONF states that the traditional hierarchical networks are well-suited for the client-server computing, but changing traffic patterns and cloud services call for a new and more flexible solution.

The key characteristic of the SDN network architecture is a separation of the forwarding and programmable network control, as defined by the Open Networking Foundation (2012); the business applications access the network service via an SDN controller (or control software in the control layer), which controls infrastructure network devices (in the forwarding/infrastructure layer); there are different APIs for the applications to access the controller, the controller itself accesses the network devices typically by means of the OpenFlow protocol. According to the same source, it is essential that the whole network state is centralized in the controller, which is programmable by network staff on a per-flow basis. Further details regarding the SDN architecture can be found in Open Networking Foundation (2013).

Hewlett-Packard SDN Programming

Hewlett-Packard SDN solution fulfils key aspects described in the previous chapter. To start experimenting with SDN programming with HP products, the HP SDN Dev Center (accessible at sdndevcenter.hp.com) provides networking staff with a downloadable SDN controller in a form of a virtual machine, with a development suite consisting of a developer virtual machine, APIs, documentation, etc.

The central part of the HP SDN solution is the SDN controller. According to HewlettPackard (2014a), the SDN controller can be accessed by external applications with the RESTful HTTPS interface, or applications can run internally (natively) within the controller – via the Java API. This paper focuses on the first option.

The REST API specification can be found in Hewlett-Packard (2014b), Kubica (2014) represents a step by step guide how to start programming SDN applications based on the REST API and Python programming language. According to these sources, the basic idea is the stateless nature of HTTP communication between the external application and the controller, i.e. use the HTTP GET method to obtain some information from the controller/network, or use the HTTP POST method to write some information to the controller/network; the data being exchanged between the application and the controller is formatted using the JSON – JavaScript Object Notation.

The HP SDN Client Python library (accessible online, incl. documentation) created by Tucker (2014) makes the interaction between Python applications and the HP SDN controller easy, and besides that it provides authentication, error handling, and Python/JSON object mutual serialization.

From the practical point of view, the ability to define a specific traffic path for a given flow on the SDN controller seems to be a promising feature. According to Kubica (2014), this task is accomplished with the OpenFlow Flow-mod message, which is comprised of a match part classifying the traffic, and an action/instruction part telling how to handle the traffic (e.g. where to send the traffic); the content of the message is written to an OpenFlow-enabled switch identified with a Data Path ID (DPID). Other information pertaining to the Flow-mod message (e.g. priority, timeouts, etc.) can be found in the same source.

First Experimental SDN Applications

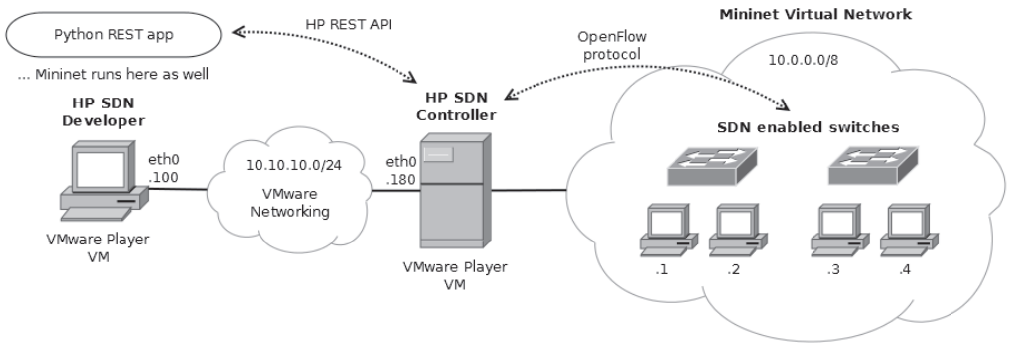
Our primary goal was to start learning SDN programming. The most effective learning path from our point of view was to begin with Kubica (2014) and test all the labs presented in this study material with the HP developer suite and with the HP SDN controller – both running as virtual machines on a PC and with a network simulated in the Mininet virtual network – details in Mininet (2014). As supportive resources we would recommend Tucker (2014) and Hewlett-Packard (2014b).

Mainly for the educational purposes, we present one simple application named "Access Layer Viewer", which closely follows Kubica (2014) – the application "Koncové stanice v GUI" on p. 31 and 32, including other pages explaining all the necessary prerequisites. We took the idea and on many places the exact programming code as well, our application then extends T. Kubica's code in that it graphically assigns end hosts to their corresponding switches, and both the switches and the hosts are depicted with usual network icons.

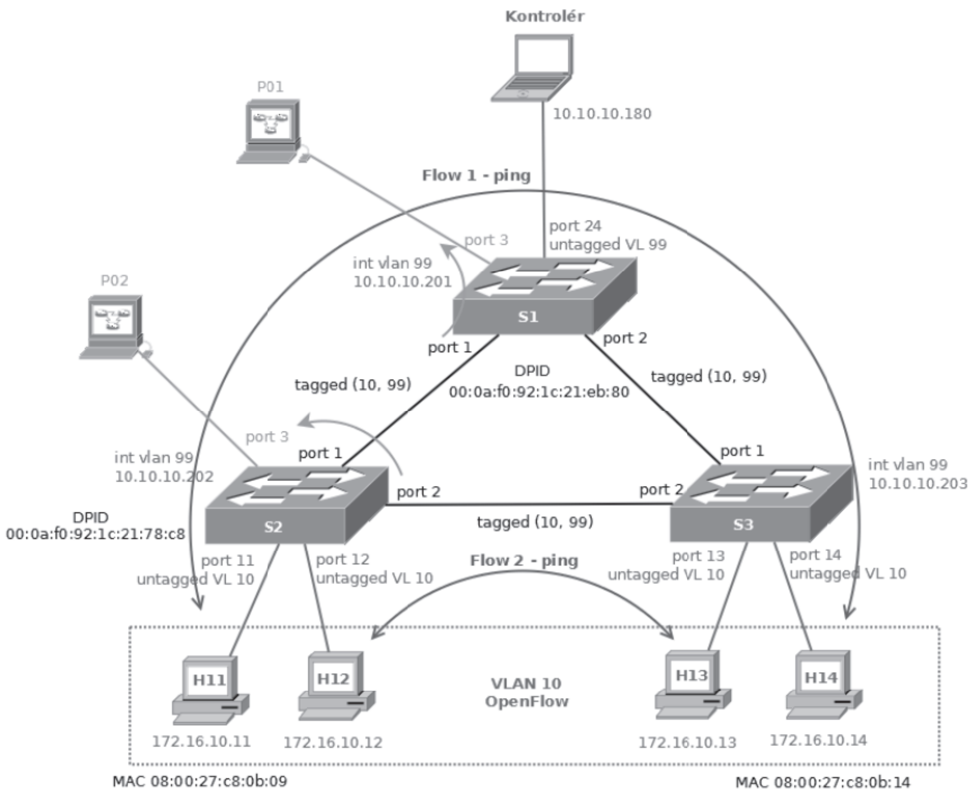
After that we wanted to perform an experiment with real network switches. Therefore, our second test case was dedicated to the problem of the traffic path definition in a laboratory network with three real OpenFlow-enabled HP switches. According to Kubica (2014), we used the HP SDN controller's web interface to access its REST API (p. 13), assembled the authentication JSON (p. 14), authenticated our web browser window (p. 15), assembled the Flow-mod JSON (p. 18 and p. 22), and uploaded this traffic path rule to the controller as a new datapath flow (p. 23). Further details are presented lower in section 2.3.2 and 3.2. While Kubica's (2014) flow definition goal is to deny traffic for a specific host, we used a modified rule to direct traffic out of a specific switch port.

Access Layer Viewer (Virtual Network)

The application connects to the controller, authenticates itself, based on controller records it discovers all end hosts connected to all access switches, and after that displays all the hosts and their corresponding switches in the application GUI. Each host is supplied with its IP address, MAC address, and switch port number. The whole test case is shown on Fig. 1.



1: *Test case of the Access Layer Viewer SDN application*



2: *Test case of the traffic path definition*

Traffic Path Definition (Real Network)

There were four end hosts in the network topology depicted on Fig. 2. All of the hosts are members of the VLAN 10 with spanning tree protocol intentionally disabled to leave the SDN controller to direct traffic inside the VLAN. The management VLAN 99 served as a dedicated VLAN for the OpenFlow communication between the controller and the switches. The default traffic path between H12 and H13 (i.e. flow 2 on the picture) follows the one-hop direct path between the switches S2 and S3. On the other hand, the traffic path between H11 and H14 (i.e. flow 1) was defined with the Flow-mod so that it takes the longer two-hop path via the switch S1 in the H11 to H14 direction. To prove the real traffic path in the network, the switch traffic mirroring feature and two monitoring hosts P01 and P02 with Wireshark were deployed.

RESULTS

Results of both of the test cases are presented in this chapter. As for the Access Layer Viewer, the application code is explained including the user's GUI window. In the second case, the traffic path definitions are described.

Access Layer Viewer – Code and GUI

The application source code with brief explanations is presented below, detailed explanations can be found in Kubica (2014). The user's GUI window is shown on Fig. 3.

1. Python script, authors, importing necessary libraries – HP SDN Client REST API and application GUI implemented with the Tkinter Python GUI library.

```
#!/usr/bin/env python

# ------------------------------------
# Experimental educational SDN application Access Layer Viewer
# Author: Martin Pokorny, Department of Informatics FBE MENDELU, 2014/2015
# -------------------------------------
# Based on:
#
# The application closely follows Tomas Kubica's guide
# "Vyvoj aplikaci pro HP VAN SDN kontroler",
# "Cast prvni - REST rozhrani a Python", version 1.02, 2014.
# Accessible at: http://www.netsvet.cz/cs/download/hp-sdn-python-lab-1.02.pdf
#
# The idea of the application and the exact programming code
# on many places below come from the T. Kubica's guide – especially
# the application "Koncove stanice v GUI" on p. 31-32 is the key source,
# including other pages explaining all the necessary prerequisites.
#
# Our contribution:
# Access Layer Viewer graphically assigns end hosts to their corresponding
# switches, and both the switches and the hosts are depicted with
# usual network icons.
# -------------------------------------

# HP SDN Client Python library, by D. Tucker (Hewlett Packard)
# Accessible at: http://hp-sdn-client.readthedocs.org/en/latest/
import hpsdnclient as hp

# Python GUI package
from Tkinter import *
```

2. Controller access and authentication of our application, GUI access.

```
# Controller authentication and connection
auth = hp.XAuthToken(user='sdn',password ='PASS',server='10.10.10.180')
api = hp.Api(controller='10.10.10.180', auth=auth)

# GUI
master = Tk()
```

3. Start of the application logic: getting list of nodes from the SDN controller.

```
# Get list of end hosts
nodes = api.get_nodes()
```

4. Our contribution starts here (still mix of our and T. Kubica's code): variables initialization, switch and the host icons come from the Dia drawing tool (available under the GPL license at http://live.gnome.org/Dia).

```
# Icons
img_sw = PhotoImage(file="sw.gif")
img_pc = PhotoImage(file="pc.gif")

# Column number for each switch (key = switch's DPID)
sw_col = {}

# Row number for actually processed host under a particular switch (key = switch's DPID)
sw_row = {}

# First column in the grid
gui_act_col = 0
```

5. Iteration for all hosts: if the host's switch is a new one (i.e. not processed yet), take another column on the right in the grid, store the column number of the actual switch into the hash "sw_col" for future use, and fill up the rows with the switch and with the first host encountered on the switch.

```
# For each end host
for node in nodes:

  # A new switch found
  if node.dpid not in sw_col:

    # First switch starts at column 0, another switches +1 to the right
    sw_col[node.dpid] = gui_act_col
    gui_act_col = gui_act_col + 1

    # First two fixed rows 0 and 1 (switch icon, switch’s DPID)
    Label(master,image=img_sw).grid(row=0,column=sw_col[node.dpid],padx=10)
    Label(master,text=node.dpid).grid(row=1,column=sw_col[node.dpid],padx=10)

    # First end host starts at row 2
    sw_row[node.dpid] = 2

    # Four pieces of information for the first host under this switch
    # (port, PC icon, IP address, MAC address)
    Label(master,text=”port “ + str(node.port)).
     grid(row=sw_row[node.dpid],column=sw_col[node.dpid],padx=10,pady=10)
    Label(master,image=img_pc).
     grid(row=sw_row[node.dpid] + 1,column=sw_col[node.dpid],padx=10)
    Label(master,text=node.ip).
     grid(row=sw_row[node.dpid] + 2,column=sw_col[node.dpid],padx=10)
    Label(master,text=node.mac).
     grid(row=sw_row[node.dpid] + 3,column=sw_col[node.dpid],padx=10)
```

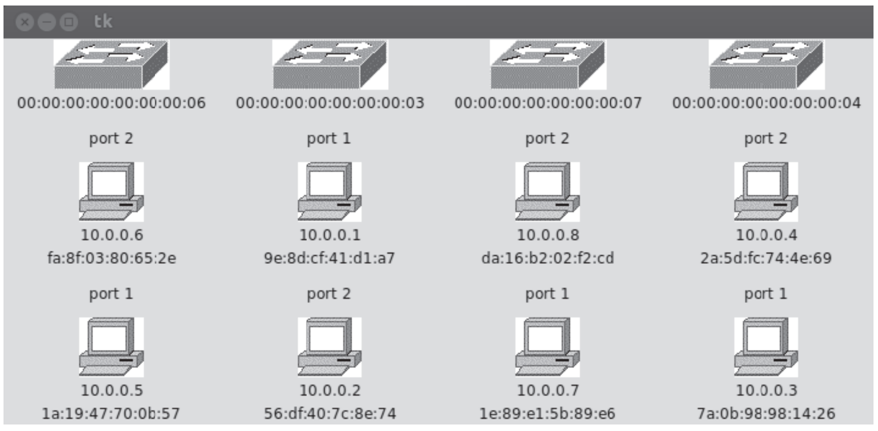
6. If the host’s switch has been already encountered, another host will be placed into the already existing switch’s column (based on column number stored in the “sw_col” hash). The host will be placed into the rows under the last host whose actual row number has been stored in the “sw_row” hash.

```
# An existing switch found
  else:

    # Another host starts four lines below the previous host
    sw_row[node.dpid] = sw_row[node.dpid] + 4

    # Four pieces of information per host (port, PC icon, IP address, MAC address)
    Label(master,text=”port “ + str(node.port)).
     grid(row=sw_row[node.dpid],column=sw_col[node.dpid],padx=10,pady=10)
    Label(master,image=img_pc).
     grid(row=sw_row[node.dpid] + 1,column=sw_col[node.dpid],padx=10)
    Label(master,text=node.ip).
     grid(row=sw_row[node.dpid] + 2,column=sw_col[node.dpid],padx=10)
    Label(master,text=node.mac).
     grid(row=sw_row[node.dpid] + 3,column=sw_col[node.dpid],padx=10)


mainloop()
```



```
00:00:00:00:00:00:00:06   00:00:00:00:00:00:00:03   00:00:00:00:00:00:00:07   00:00:00:00:00:00:00:04
        port 2                    port 1                    port 2                    port 2
       10.0.0.6                  10.0.0.1                  10.0.0.8                  10.0.0.4
  fa:8f:03:80:65:2e         9e:8d:cf:41:d1:a7         da:16:b2:02:f2:cd         2a:5d:fc:74:4e:69
        port 1                    port 2                    port 1                    port 1
       10.0.0.5                  10.0.0.2                  10.0.0.7                  10.0.0.3
  1a:19:47:70:0b:57         56:df:40:7c:8e:74         1e:89:e1:5b:89:e6         7a:0b:98:98:14:26
```

3: *Access Layer Viewer SDN application screenshot*

Traffic Path Definition

The default path between hosts H11 and H14 is the same as the path between H12 and H13, i.e. the shorter one-hop path via switches S2 and S3. To redefine the forward path from H11 to H14 to go via the upper switch S1, two Flow-mod JSONs were created – one for the switch S2 and the second one for the switch S1 (shown below). The match section identifies both of the end hosts with their MAC addresses, the actions section informs the switch about an output port number, i.e. where to forward matching packets. Priority 31 000 is higher than the default priority of the shorter path (29 999).

S2 (DPID 00:0a:f0:92:1c:21:78:c8) Flow-mod JSON:

```
{"flow": {
  "priority": 31000,
  "hard_timeout": 30,
  "match": [
    {"eth_type": "ipv4"},
    {"eth_src": "08:00:27:c8:0b:09"},
    {"eth_dst": "08:00:27:c8:0b:14"}
  ],
    "actions": [{"output": 1}]
  }
}
```

S1 (DPID 00:0a:f0:92:1c:21:eb:80) Flow-mod JSON:

```
{"flow": {
  "priority": 31000,
  "hard_timeout": 30,
  "match": [
    {"eth_type": "ipv4"},
    {"eth_src": "08:00:27:c8:0b:09"},
    {"eth_dst": "08:00:27:c8:0b:14"}
  ],
    "actions": [{"output": 2}]
  }
}
```

The definitions above ensure only the forward path from the source to the destination host (i.e. ping Echo Request along the path H11–S2–S1–S3–H14), but not the return path (Echo Reply) that follows the original one-hop path (H14–S3–S2–H11). To make the return path the same, another traffic path definitions would be necessary.

The switch configuration concerning VLAN, ports, mirroring, STP, and OpenFlow is not shown here for brevity – the configuration commands can be found in manuals available with the switch product line used.

DISCUSSION AND CONCLUSION

Management of today's networks can get out of hand with a large number of network devices. There are sophisticated network management tools providing centralized device configuration, image updates, and system backup, nevertheless, this way of network maintenance still requires one by one device access logic, many networking protocols being deployed on each device, and all these usually pose high costs. The SDN seems to be a promising way how to reduce the costs, because much of the distributed network logic can be centralized into a single device, the SDN controller. We anticipate that the SDN approach can change network design and implementation in near future, similarly as the WiFi controllers changed the way how to control a large number of WiFi access points. In our opinion, it is now critical for the major network vendors to finish OpenFlow implementation in their products, and start educational process in the network community.

In our paper, we selected a few useful sources of documentation to start learning SDN. As for the implementation, we chose the Hewlett-Packard solution, and according to HP's manuals and guidelines, we performed two experiments: firstly we extended T. Kubica's SDN application providing information about end hosts in the network (the Access Layer Viewer application), and after that, we performed an experiment of traffic path definition with real network switches. As a further work, we would like to incorporate full network topology visualization and link utilization statistics to the Access Layer Viewer.

Acknowledgement

We would like to thank Hewlett-Packard for providing us with SDN demonstration, study guides and test bundle with OpenFlow enabled switches, which helped us to start experimenting with SDN.

REFERENCES

HEWLETT-PACKARD. 2014a. *HP VAN SDN Controller Software. Data sheet*. [Online]. Available at: http://h20195.www2.hp.com/v2/getpdf.aspx/4AA4-9827ENW.pdf. [Accessed: 2015, February 11].

HEWLETT-PACKARD. 2014b. *HP VAN SDN Controller 2.3 REST API Reference*. [Online]. Available at: http://h20564.www2.hp.com/hpsc/doc/public/display?docId=c04383855. [Accessed: 2015, February 11].

KUBICA, T. 2014. *Vývoj aplikací pro HP VAN SDN kontroler. Část první – REST rozhraní a Python. Ver. 1.02*. [Online]. Available at: http://www.netsvet.cz/cs/download/hp-sdn-python-lab-1.02.pdf. [Accessed: 2015, February 10].

MININET. 2014. *Mininet: An Instant Virtual Network on your Laptop (or other PC).* [Online]. Available at: http://mininet.org. [Accessed: 2015, February 10].

OPEN NETWORKING FOUNDATION. 2012. *Software-Defined Networking: The New Norm for Networks. ONF White Paper*. [Online]. Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Accessed: 2015, February 9].

OPEN NETWORKING FOUNDATION. 2013. *SDN Architecture Overview*. [Online]. Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf. [Accessed: 2015, February 9].

TUCKER, D. 2014. *HP SDN Client: A Python library that makes interaction with the HP VAN SDN Controller REST API easy*. Hewlett-Packard. Release v1.1.0. [Online]. Available at: http://hp-sdn-client.readthedocs.org/en/latest/. [Accessed: 2015, February 11].

Contact information

Martin Pokorný: martin.pokorny@mendelu.cz
Petr Zach: petr.zach@mendelu.cz