In-band distributed SDN Control Plane with Sample Application

Klajd Agolli, Syed Emad, Sehrish Fatima
Supervisor: Rhaban Hark
Multimedia Communications Lab II (TuCan ID 18-sm-2070)
Technische Universität Darmstadt
Darmstadt, Germany

Abstract -- **Nowadays Software-Defined Networking (SDN) has found its way into commercial applications, however many questions regarding its performance, scalability and availability have not yet been answered definitively. In SDN, the controller plane is decoupled from the network hardware devices. It is logically centralized with a global view of the entire network topology. The network devices such as switches and routers (the SDN-data plane) are responsible for processing and forwarding packets according to the installed rules. OpenFlow (herein OF) is currently the most commonly deployed SDN Southbound Interface. Even if SDN obtains an important part of its advantages through the use of logically centralized controller with global view of the network dynamic state, the fact that the availability of the platform would depend on a single physical controller at a given time, would create scalability and availability problems. For this reason, there is a wide consensus that the control plane should be physically distributed. Physically distributed SDN control planes ensure high availability and fault tolerance. But of course in such a case, different controller instances need to be synchronized / to communicate in order to guarantee the consistency of the network operations, the correct implementation of the high-level policies (e.g. endpoint-, routing-policies provided by the network designers), and to enable optimal usage of the available data plane resources. The main focus and contribution of this lab is to design and implement an in-band control-plane communication layer, which provides connectivity and communication between controllers and switches, as well as among the controllers themselves. Controller discovery and isolation of controller traffic are also required features. The implemented controller module establishes automatically for each new SDN controller a communication channel to all the other available controller instances, provide a secure tunneling of the controller information data-plane switches, enabling so a dynamic and self-organized distributed SDN control plane.**

Keywords
Control Plane; Communication Layer; Data Plane; Distributed System; In-band; OpenFlow; Software-Defined Networking; Southbound Interface; Virtual Network.

I. INTRODUCTION

SDN represents a novel and innovative idea in the networking domain. It proposes a new approach of decoupling the control plane from the data plane of individual network components, where a logically centralized control-plane with a global view of the entire network topology are responsible for the configuration and management of any network device in the data plane. On the other side, the network devices or the data plane is responsible only for processing and forwarding packets according to the installed rules. They also collect statistical traffic state information for the controller.

During the last years, SDN in general and OpenFlow (the most commonly deployed SDN technology [1, 8, 10]) in particular has been widely deployed in real-life large-scale networks. Switches and routers that support SDN/OpenFlow are provided by a large number of vendors [3]. However, there are different aspects in the SDN architecture, which may create performance, scalability and availability problems for the deployed network applications. A typical example was mentioned previously, namely the necessity for a logically centralized, but physically distributed SDN controller plane. The separation of the software-defined networks into "dumb" network devices/data plane and "smart" logically centralized control plane with a network-wide view represents the essence of the flexibility in SDN [14]. But, the fact that the availability of the platform would depend on a single physical controller at a given time, would create scalability and availability problems [1]. For this reason, there is a wide consensus that the control-plane should be physically distributed in order to [12, 15. 16]:

- Provide high availability and fault tolerance.
- Handle latency-sensitive and communication-intensive data plane events close to their origin.
- Distribute the management responsibilities and overheads (load balancing).

However, different controllers within the SDN control-plane need to be synchronized, to communicate with each other in order to:

- Guarantee consistency of network operations.
- Enable optimal usage of the available data plane resources

So the main task and contribute of this Lab is to design and implement an in-band control-plane communication layer that provide the following required services:

- Connectivity, in order to allow communication between controllers and switches, and between controllers.
- Controller discovery, to enable individual controllers to discover the existence of other controllers and to detect when some are no longer reachable.
- Security, by providing isolation of the controller traffic through the data plane switches.
- Switch discovery, by enabling the controllers to detect and establish a control channel also with unmanaged, not directly connected switches.

The implemented Floodlight controller module establishes automatically for each new SDN controller a communication channel to all the other available controller instances, provides a secure tunneling of the controller information data-plane switches, enabling so a dynamic and self-organized distributed SDN control plane.

The rest of this paper proceeds as follows. In Section II (state of the art) we provide a brief introduction of SDN and OpenFlow in general, explain why do we need a physically distributed SDN control-plane, and of course also the different approaches how to implement it. Section III (design and implementation) presents our approach regarding the design of the distributed control-plane communication layer, the frameworks and tools we have used in order to simulate the SDN environment and the concrete implementation of the required Floodlight module. Section V (discussion and conclusion) gives some concluding remarks for future works.

II. State of the Art / Related Work

SDN is a recent innovative paradigm in the networking domain. It proposes a new approach of decoupling the control plane from the data plane of individual network components [10], where a logically centralized controller with a global view of the entire network topology is responsible for the configuration and management of any network device in the data plane. The network controller can be considered as an art of compiler that translates the abstract high-level policy (endpoint policy [17]) and lower-level routing policy [17] provided by the network designers into specific switch-level rules to be stored on the respective flow-table of the data plane switches [3]. Otherwise, the data plane network devices are relatively simple ("dumb" network devices/data plane). Their main purpose is to process and forward the packets according to the installed rules, as well as to monitor local events and to collect statistical traffic state information for the controller. Each switch contains several consecutive matching tables, each of them having installed a strictly prioritized list of rules [6]. The rules in the flow-tables consist of at least two fields, one packet header matching field and one action field [6, 13]. The matching field contains an array of ternary elements (0,1, *), which specifies how the packet header will be matched against this rule. The action field specifies the action to be executed on the matched packets (e.g. drop the packet, forward it, or increment counters for specific measurements) [6, 13]. The action field may also include packet header modification, such as changing the VLAN tag or TTL field [9]. It is clear that SDN switch's tables are dynamic entities, which evolve depending on the traffic load across the respective device, and the controller is responsible for the management of the network devices by adding and removing the rules in the matching tables through special messages. In its simplest implementation SDN infrastructure work more or less like this [4]: Each network device directs the first packet of a new flow to the controller, which reactively install an exact-match rule on the switch table (not really efficient in terms of controller overhead and data plane resources usage). In addition switches can also send messages to controller (e.g. measurements taken in the built-in counters, or notification regarding local events).
There are many advantages of using SDN:

- More flexibility in network services innovations. Its key concept of managing the entire network as a unified abstraction, and the remote control of the network devices through open protocols, such as OpenFlow, provides new flexibility in network innovation [2]
- Simplify the administration of the network platform. It enables much easier network configuration as well as easier development of new network protocols and polices [3].
- Reduce costs for the network devices. Due to the concept of "dumb" network devices/data plane, the SDN switches are functionally simple, easy to design and so even with lower production costs [6].
- Enable efficient handling for diverse traffic patterns. By applying fine-grained packet processing rules that match on multiple header fields, SDN enables a wide range of applications such as firewalls, load balancers, routers, traffic monitoring, and other functionalities [5]

SND has become an important paradigm in the todays computing networks and OpenFlow [7] is currently the most commonly deployed SDN southbound interface [1, 8, 10]. It was proposed as a solution to standardize the communication between controllers and network devices in the SDN architecture [1]. OpenFlow specifications (e.g. in [7, 9]) define a flow-based forwarding approach and its architecture consists of the following three basic concepts [1, 8]:

- "OpenFlow-compliant switches" [8] that compose the data plane.
- One or more OpenFlow controllers composing the control plane
- A secure control channel that connects the switches with the control plane.

OpenFlow-enabled switches are basic network devices, responsible for classifying packets and performing actions on them according to their flow table [8]. The table holds a set of flow entries also called flow rules, which is evaluated on every packet entry. Each rule in the table is composed of three fields [8]: the matching/header fields, the counters fields and the action fields. The matching field in a flow entry determines the packets to which the entry is practicable [8]. Each flow rule can match on multiple packet header fields (from layer 2 to layer 4 of the current ISO/OSI Network Layer Model [10]) depending on the deployed application. “For example, an access-control application may match on the traditional flow-key tuple (source and destination IP-Address, transport protocol, source and destination port number), while a server load balancing application on the destination IP address and the source IP prefix” [5]. Traffic can be matched against fine-grained matching rules (microflow [4]) or more general rules using wildcard bits. The counters fields are reserved for collecting statistics about flows (number of packets, bytes, as well as the flow-duration) [8]. The information collected from the three counters is exposed to the central controller. The actions field specifies how packets of the matching flow/flows should be handled (common actions are: drop the packet, forward it and/or modify header fields, etc) [8, 1].

The controller is a software program responsible for populating and managing the flow tables of the switches [8]. By adding and removing flow entries, the controller implements the behavior of the switches with regard to packet processing. As mentioned previously, even if SDN obtains an important part of its advantages through the use of a logically centralized control-plane with a global view of the network dynamic state [14], the fact that the availability of the platform would depend on a single controller at a given time, would create scalability and availability problems [1]. Frequent and resource-exhaustive events, such as new flow arrivals (installation of new entries on the switches’ flow table) or network-wide statistics collection events (getting the information storied in the in-built counters) stress the controller and consequently limit the scalability of the OpenFlow-based networking applications [18]. The available bandwidth between controller and the switches (the control channel bandwidth [10]) is a limited resource, as well as the processing capacities of the device hosting the controller software program. So, especially in case of an extensive use of reactive flow setup, “such resources limitations.

For all this reasons, there is a wide consensus that the control-plane should be physically distributed, and there are generally speaking there are two alternatives how to implement such a SDN control-Plane, namely, **Out-of-band** and **In-band** SDN control plane:

Out-of-Band: The control network is physically separated from the data network and the controllers never tunnel control information through the data network. Such a distributed SDN control-plane, is much simpler to realize and in some cases [16] is also considered as a more resilient solution compared to the in-band alternative. Of course the most important disadvantage of the out-of-band alternative is the fact, that it requires dedicated control network, increasing so costs and the network complexity.

In-Band: In-band distributed SDN control plane use the existing data plane technology to enable the communication between the available controller instances. The control information is tunneled through the data-plane switches. For these reason the in-band solution:

- requires no dedicated control network required.
- enables functional control-plane as as long as the underlying data plane is also OK.
- enables a dynamic control plane.

Of course, it requires efficient controller discovery approaches, and the isolation of the control traffic with the data plane must be guaranteed.

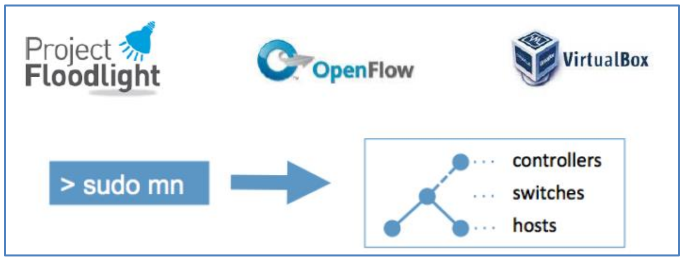
Medieval [12], one of the in-band SDN control network applications, endorses the distributed control plane service. Its main objective is to have one switch under the control of a single controller. It discovers topology automatically and provides connectivity between the controllers as well as dynamically adding or removing controllers. Every controller constantly runs the algorithm to monitor any change in the network such as addition or removal of a switch, route or a controller. Medieval creates and maintains two specific spanning trees for every controller:

1-A per-region spanning tree
2-A network-wide spanning tree

At the beginning, the initial region of a controller only comprises of the controller itself. Next, the directly connected switches of the controller are added to the region. Next, 1-hop away switches are added to the region of the controller, and so on. Each switch is pre-configured with a virtual controller IP and a set of OpenFlow rules, applicable to the control packets. When the controller starts an OpenFlow session with the switch, the controller advances to install the ownership rules into the connected switch first.

III. Our Approach / Design and Implementation

For implementation of the task, the networking tools and frameworks utilized are discussed below:



FLOODLIGHT [19]:

Floodlight is an open source implementation of SDN Controller written in java. It communicates with the physical and virtual switches using the Openflow protocol. Floodlight is designed to work with the growing number of switches, routers, virtual switches, and access points that support the OpenFlow standard. Floodlight is built upon a modular structure which enables programmers to add modules and export services to be used by other modules. The controller framework enables the following basic functions:

- Link discovery and topology manger
- Install/Modify/Delete of the forwarding rules on the switches flow-tables
- Provides to the controller statistical traffic information regarding the respective data-plane domains.

MININET [20]:

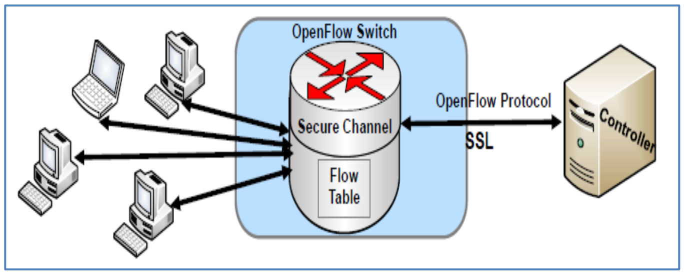
Mininet is an Open source networking tool to simulate the virtual OpenFlow-compliant network. It is used to create virtual networks, running real kernel, switch and application code, on a single machine (VM, cloud or native), Mininet is useful for development, teaching and research purposes since it can be easily customized, shared or deployed on real hardware. We used to Mininet to create and launch the network configuration shown figure 3 below. The network configuration is written as a python script stored in the Mininet VM.

VIRTUALBOX [21]:

Virtualbox is a cross platform virtualization application to provide the required infrastructure. It extends the capabilities of your existing computer so that it can run multiple operating systems (inside multiple virtual machines) at the same time.

OPENFLOW v1.3 [1, 8, 10]:

Openflow is a communication protocol that gives direct access to and manipulation of the forwarding plane of a network switch or router over the network (both physical and virtual, hypervisor-based) [11]. SDN Southbound Interface to enable the interaction between the controller and the OpenFlow-compliant switches.

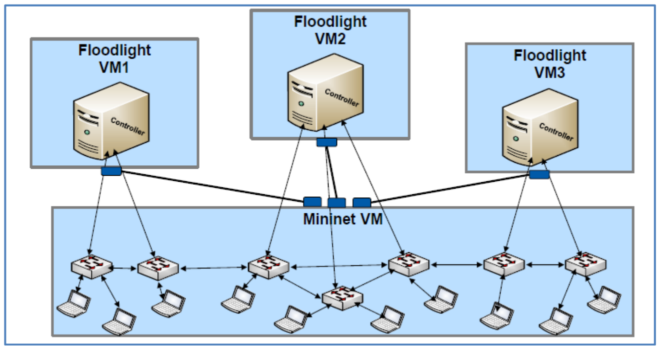


It allows programmability by introduction of new services and enables innovation. In traditional routers or switches the two fundamental functions of routing and forwarding reside in the same device but openflow protocol enables moving the routing logic to an external controller which then communicates the routing instructions to the switches via the openflow protocol.

An OpenFlow switch consists of one or more flow tables and a group table, which carry on packet lookups and forwarding. There is an OpenFlow channel to an external controller which manages the switch via the OpenFlow protocol.

The aim of the implemented Floodlight module is to achieve implementation of controller discovery and to enable controllers to communicate with each other. This will create an extension of floodlight & existing OpenFlow. The controller broadcasts messages to other controllers. After receiving a broadcast message, the receiving controller installs a modification of rule (with port number or IP address), sends it back to the initial controller. We provide the functionality of this module as a service for meaningful use.

The network configuration is explained in the Figure below:



A detailed description of the working of this module is provided below.

The network python script in the mininet VM is hardcoded with IP address of a controller. This controller facilitates the communication between different hosts in the network allowing them to contact one another.

Each time that a new switch is added to a controller e.g. controller1, it sends out a “Controller Discovery Packet” through the network. Controller1 extracts the DatapathID and Mac address of the switch. Controller1 then creates a new broadcast packet which includes this information about the switch in its payload. The packet is then broadcast through that switch using the Packet_out message using 65003 and 67 as transport source and destination ports respectively.

When one of the other controllers (e.g. Controller2) receives the Controller Discovery packet, it extracts the information from this packet and first checks its own list of already known controllers to see if the controller sending this request is already known to it. If the IP Address of Controller1 is not in that list then, Controller2 adds the Controller1 IP address and the ingress port in its lists. The controller also lists the switch, from which it received the packet_in message as a Gateway Switch for packets directed to Controller.

As a response to the controller discovery request, controller2 then creates a new controller discovery replay packet using the source IP address from the controller discovery request packet as the destination address and setting the source and destination ports as 65004 and 67 respectively.

Controller2 follows it up by formulating and installing a flow modification rule in order to handle any further packets from controller1. This is done by using the flow mod message of the openflow protocol. In this process controller2 uses the switch ingress port of the controller discovery request message as the switch output port through which all future packets destined to controller1 have to be forwarded.

When Controller 1 receives the "Controller Discovery Replay", it does the same with the required information and installs a new flow-rule on its gateway switch.

Any time that one of the switches contained in the list of Gateway Switches is removed; the controller removes from its list of already known controllers, the IPs of all the controllers associated with that gateway switch and starts again the Controller Discovery Process especially to find a new route for the removed controllers but in this case the created packet uses 65005 as the transport source port and 67 as transport destination port.

When a controller receives the controller discovery packet resulting from switch removal, it sends back a controller discovery replay packet back using 65006 and 67 as the transport source and destination ports resp. The controller also installs the flow modification rule into the gateway switch in a similar manner as mentioned above. The controller receiving the discovery replay packet after switch removal responds by updating its lists. This mechanism ensures connectivity between the controllers.

In order for other modules to benefit from our implementation we export the created lists as a service. This is done by creating an interface which extends the IFloodlightService interface. This is a standard way for exporting services in Floodlight. In our interface we provide getter methods which enable other modules to acquire the lists containing controller IP addresses, controller gateway switches, the gateway switch ports and the gateway switch mac addresses.

The entire algorithm is summed up as follows:

```
1. Switch Added:

   • Create a new "Controller Discovery Request " packet.
   • Broadcast the packet using the new added switch as Gateway .

2. "Controller Discovery Request" received:

      If (srcIP  not found in IP list of already known controller):
      ➢ Save the following information:
            • srcIP as newController IP
            • srcMac as the Mac Address of the newController
            • Switch as the GatewaySwitch for the newController
            • Switch IN_PORT as the GatewayPort for the newController

      ➢ Send "Controller Discovery Replay " packet with srcIP as destination IP.
      ➢ Install flow-mod rule in the GatewaySwitch:
            • MatchField: Destination Ip = srcIp
            • Action: Forward Output_Port = GatewayPort

3. "Controller Discovery Replay" received:

      If (srcIP  not found in IP list of already known controller):
      ➢ Save the following information:
            • srcIP as newController IP
            • srcMac as the Mac Address of the newController
            • Switch as the GatewaySwitch for the newController
            • Switch IN_PORT as the GatewayPort for the newController
      ➢ Install flow-mod rule in the GatewaySwitch:
            • MatchField: Destination Ip = srcIp
            • Action: Forward Output_Port = GatewayPort

4. One of the Gateway Switches Removed:

   ➢ Remove IP address of all controllers with this switch as GatewaySwitch from list.
   ➢ Remove switch mac addresses from list.
   ➢ Remove switch port numbers from list.
   ➢ Create a new "Controller Discovery Request" packet for each of the removed controller.
   ➢ Broadcast the packet to find an alternative path

5. "Controller Discovery Request" after switch removal received:

      ➢ Send "Controller Discovery Replay " packet with srcIP as destination IP.

6. "Controller Discovery Replay" after switch removal received:
      If (srcIP  not found in IP list of already known controller):
      ➢ Save the following information:
            • srcIP as newController IP
            • srcMac as the Mac Address of the newController
            • Switch as the GatewaySwitch for the newController
            • Switch IN_PORT as the GatewayPort for the newController
      ➢ Install flow-mod rule in the GatewaySwitch:
            • MatchField: Destination Ip = srcIp
            • Action: Forward Output_Port = GatewayPort
```

IV. Discussion and Conclusion

Our implementation is meant to be an extension to the floodlight project. A module which aims to provide discovery and communication service between OpenFlow controllers in a network using the existing data-plane network infrastructure. We believe that this approach is simple and circumvents any unnecessary obstacles associated with out of band methodologies. With our work we strive to make a contribution to the ongoing research in the field of software defined networking with particular emphasis on in-band distributed control planes. The implemented floodlight module provide the required controller discovery service, enabling individual controllers to discover the existence of other controllers and to detect when some are no longer reachable, and it provides for each of the controller the required communication channel to all the other available controller instances, enabling so a dynamic and self-organized distributed SDN control plane.

Security: The security of the control-plane information through the data plane was also an important requirement for the required module. In the actual implementation we have try to provide some isolation of the controller within the data-plane by specifying dedicated tagged VLANs for the control-plane packets. Of course we have to say, that using tagged VLANs does not really satisfy our initial security requirements, but a port-based VLAN (which could be a better solution) was, due to the actual floodlight topology and link discovery service, not possible to be implemented.

Efficiency: The actual solution is pretty simple, and it is not really efficient in term of controller discovery traffic overhead, as well as the provided communication channel between two controllers.

Efficiency and security are both topics to be considered in future works. In our future work we plan to enhance this module by including functionalities for switch discovery and acquiring network topology information with the ultimate goal of integrating it into the Floodlight openflow controller.

References

[1] A. Lara; A. Kolasani; B. Ramamurthy, "Network Innovation Using OpenFlow: A Survey". IEEE Communications Surveys&Tutorials. 2013, 16, 1–20.

[2] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. TranGia, "Modeling and performance evaluation of an openflow architecture". In Teletraffic Congress (ITC), September 2011 23rd International, pages 1–7.

[3] Y. Kanizo, D. Hay, I. Keslassy. "Palette: Distributing tables in software-defined networks". In INFOCOM, pages 545–549, 2013.

[4] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, J. Rexford, "Efficient Traffic Splitting on Commodity Switches". In ACM SIGCOMM CoNEXT Conference, December 2015.

[5] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. Infinite CacheFlow in Software-Defined Networks, in the ACM SIGCOMM HotSDN Workshop, August 2014

[6] Sh. Zhang, F. Ivancic, C. Lumezanu, Y. Yuan, A. Gupta, Sharad Malik. An Adaptable Rule Placement for Software-Defined Networks, DSN 2014.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling Innovation in

Campus Networks”. ACM SIGCOMM Comput. Commun. Rev. 2008, 38, 69–74.

[8] W. Braun, M. Menth, “Software-Defined Networking Using OpenFlow”. Future Internet, 2014.

[9] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.1.0. 2011. Available online: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf (accessed on 25 Januar 2016).

[10] S. Bleinder, “Identification and Design of a Data Plane Resource Optimization Mechanisms for Application-Controlled SDN”. Master Thesis, Technische Universität Darmstadt, Department of Electrical Engineering and Information Technology Institute of Data Technology, may 2015.

[11] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

[12] S. Liron, S. Schmid, M. Canini;. "Medieval: Towards A Self-Stabilizing, Plug & Play, In-Band SDN Control Network." SOSR (Demo) (2015).

[13] Flowgrammable, Driving the Next SDN Genetration;OpenFlow 1.3.0: http://flowgrammable.org/sdn/openflow/actions/#tab_ofp_1_3, (accessed on 10.07.2016).

[14] J. C. Mogul and P. Congdon, “Hey, you darned counters! Get off my ASIC!”. First Workshop on Hot Topics in Software Defined Networks (HotSDN), 2012.

[15] S. Liron, S. Schmid, and P. Kustenzov: "In-Band Synchronization for "Distributed SDN Control Planes". ACM SIGCOMM Computer Communication Review, Volume 46, Number 1, January 2016.

[16] A. Panday, C. Scotty, A. Ghodsiy, T. Koponen, S. Shenkery; "CAP for Networks", HotSDN’13, August 16, 2013, Hong Kong, China.

[17] N. Kang, Zh. Liu, J. Rexford, D. Walker. “Optimizing the one big switch abstraction in software-defined networks”. In ACM SIGCOMM CoNext Conference, December 2013.

[18] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications”. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), 2012.

[19] Project Floodlight: http://www.projectfloodlight.org/floodlight/ , (accessed on 10.07.2016).

[20] Mininet Tool: http://mininet.org/ , (accessed on 10.07.2016).

[21] Oracle VirtualBox: https://www.virtualbox.org/ ,(accessed on 10.07.2016).