11. Diakov, V. P. (2007). Soil deformation mechanism and model the critical speed of loading. Advances in science and technology of agriculture, 10, 51–53.
12. Phan, Xuan Dung (1992). Substantiation of the parameters of the rotary cultivator working body propasnogo. Tashkent, 17.
13. Shamota, V. A. (1986). Justification shape and size of processing row spacing Garden. Kishinev : Science, 92–95.
14. Matryshko, V. M. (1999). Research results mills with vertical axis rotation Kyiv : NAU, 5, 245–250.

Рекомендовано до публікації д-р техн. наук Дмитриков В. П.
Дата надходження рукопису 28.09.2014

Прасолов Євген Якович, кандидат технічних наук, кафедра безпеки життєдіяльності, Полтавська державна аграрна академія, вул. Сковороди, 1, м. Полтава, Україна, 36003
E-mail: belovol_sa@mail.ru
Крьока Максим Вікторович, Полтавська державна аграрна академія, вул. Сковороди, 1, м. Полтава, Україна, 36003
E-mail: belovol_sa@mail.ru
Левін Володимир Валер'янович, Полтавська державна аграрна академія, вул. Сковороди, 1, м. Полтава, Україна, 36003
E-mail: belovol_sa@mail.ru

UDC 621.391
DOI: 10.15587/2313-8416.2014.27459

TESTING AND ANALYSIS SDN TECHNOLOGY

©Taher Abdullah

The Software Defined Networking (SDN) is currently one of the most promising technologies in mobile backhaul networks based on the OpenFlow protocol. OpenFlow provides a specification to migrate the control logic from a switch into the controller. In this paper we apply Mininet software to verify the OpenFlow protocol messages.
Keywords: SDN openflow protocol, switch, Mininet, Wireshark

Software Defined Networking (програмно-конфігурована мережа) (SDN) є в даний час однією з найбільш перспективних технологій в мобільних мережах транзитних з'єднань на основі протоколу OpenFlow. OpenFlow надає специфікацію для перенастроювання керуючої логіки від комутатора в контролері. У даній роботі ми застосовуємо програмне забезпечення Mininet для перевірки повідомлення протоколу OpenFlow.
Ключові слова: протокол OpenFlow програмно-конфігурованої мережі (або протокол OpenFlow SDN), комутатор, Mininet, Wireshark

1. Introduction

Software Defined Networking (SDN) is a new approach in networking Technology, designed to create high level abstractions on top of which hardware and software infra-structure can be built to support new cloud computing applications. SDN is also referred to as programmable network, since it isolates control plane from data plane and pro-vides an independent and centralized unit to control the network [1].

OpenFlow protocol follows SDN approach, and gives programmable control of flows to network administrators to define a path that a flow takes from source to destination regardless of the network topology, and utilizes flow based processing for forwarding packets. OpenFlow has gathered significant interest among developers and manufacturers of network switches, routers, and servers.

The original idea of SDN described in was born at Stanford University around 2005. The SDN concept brings the separation of network device features to the control plane, and the data plane [2]. While the control plane is programmatically accessible through well-defined API (Application Programming Inter- face), data plane ensures a data processing according to the rules uploaded to the device. OpenFlow has been developed since 2007, and the first protocol specification was approved in 2009. Lately the development was adopted by the Open Networking Foundation (ONF) consortium.

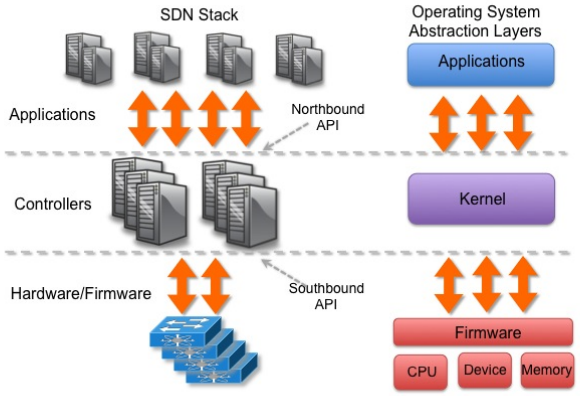


Fig. 1. A SDN approach to separate several layers and introduce transparency of the network

The protocol defines the structure of control messages and it describes the way how messages are exchanged. OpenFlow is based on the centralized approach with a controller as a main driving element. This controller runs a software platform with the API enabling the direct control of data flows in a network.

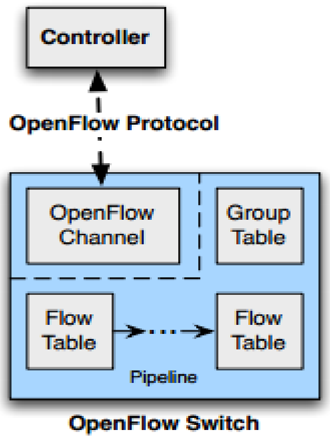


Fig. 2. Main components of an OpenFlow switch [2]

Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network. As a result, the network appears to the applications and policy engines as a single, logical switch [3]. With SDN, enterprises and carriers gain vendor independent control over the entire network from a single logical point, which greatly simplifies the network design and operation. SDN also greatly simplifies the network devices themselves, since they no longer need to understand and process thousands of protocol standards but merely accept instructions from the SDN controllers. Perhaps most importantly, network operators and administrators can programmatically configure this simplified network abstraction rather than having to hand-code tens of thousands of lines of configuration scattered among thousands of devices. In addition, leveraging the SDN controller's centralized intelligence, IT can alter network behavior in real-time and deploy new applications and network services in a matter of hours or days, rather than the weeks or months needed today. By centralizing network state in the control layer, SDN gives network managers the flexibility to configure, manage, secure, and optimize network resources via dynamic, automated SDN programs. Moreover, they can write these programs themselves and not wait for features to be embedded in vendors' proprietary and closed software environments in the middle of the network. In addition to abstracting the network, SDN architectures support a set of APIs that make it possible to implement common network services, including routing, multicast, security, access control, bandwidth management, traffic engineering, quality of service, processor and storage optimization, energy usage, and all forms of policy management, custom tailored to meet business objectives. For example, an SDN architecture makes it easy to define and enforce consistent policies across both wired and wireless connections on a campus.

2. Software-defined networking

The SDN architecture is remarkably flexible; it can operate with different types of switches and at different protocol layers [4]. SDN controllers and switches can be implemented for Ethernet switches (Layer 2), Internet routers (Layer 3), transport (Layer 4) switching, or application layer switching and routing. SDN relies on the common functions found on networking devices, which essentially involve forwarding packets based on some form of flow definition.

An OpenFlow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an Open Flow channel to an external controller [5]. The switch communicates with the controller and the controller manages the switch via the Open Flow protocol. Using the Open Flow protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to Matching packets.

First matching entry in each table is being used. If a matching entry is found, the instructions associated with the specific flow entry are executed. If no match is found in a flow table, the outcome depends on configuration of the table-miss flow entry: for example, the packet may be forwarded to the controller over the OpenFlow channel, dropped, or may continue to the next flow table. Instructions associated with each flow entry either contain actions or modify pipeline processing. Actions included in instructions describe packet forwarding, packet modification and group table processing. Pipeline processing instructions allow packets to be sent to subsequent tables for further processing and allow information, in the form of metadata, to be communicated between tables. Table pipeline processing stops when the instruction set associated with a matching flow entry does not specify a next table; at this point the packet is usually modied and forwarded. Flow entries may forward to a port. This is usually a physical port, but it may also be a logical port defined by the switch or a reserved port defined by this specification. Reserved ports may specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing , while switch-defined logical ports may specify link aggregation groups, tunnels or loopback interfaces. Actions associated with flow entries may also direct packets to a group, which specifies additional processing. Groups represent sets of actions for flooding, as well as more complex forwarding semantics (e. g. multipath, fast reroute, and link aggregation). As a general layer of indirection, groups also enable multiple flow entries to forward to a single identifier (e. g. IP forwarding to a common next hop). This abstraction allows common output actions across flow entries to be changed efficiently. The group table contains group entries; each group entry contains a list of action buckets with specific semantics dependent on group type. The actions in one or more action buckets are applied to packets sent to the group. Switch designers are free to implement the internals in any way convenient, provided that correct match and instruction semantics are preserved. For example, while a flow entry may use an all group to forward to multiple ports, a switch designer may choose to implement this as a single bitmask within the hardware forwarding table. Another example is matching; the pipeline exposed by an OpenFlow switch may be physically implemented with a number of hardware tables.

3. SDN In MININET Program

3. 1. Mininet Program

Mininet *is a* network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking [6].

Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC. Mininet provides an easy way to get correct system behavior (and, to the extent supported by your hardware, performance) and to experiment with topologies.

Mininet networks run *real code* including standard Unix/Linux network applications as well as the real Linux kernel and network stack (including any kernel extensions which you may have available, as long as they are compatible with network namespaces.) Because of this, the code you develop and test on Mininet, for an OpenFlow controller, modified switch, or host, *can move to a real system with minimal changes*, for real-world testing, performance evaluation, and deployment. Importantly this means that a design that works in Mininet can usually move directly to hardware switches for line-rate packet forwarding.

We will need to install these files individually. The files include virtualization software, a SSH-capable terminal, an X server, and the VM image. The tutorial image is distributed as a compressed VirtualBox image (vdi). VirtualBox enables you to run a virtual machine inside a physical machine, and is free and available for Windows, Mac and Linux [7].

3. 2. Message between Controller with Switch by Open flow Protocol

The communication between the controller and switch happens using the OpenFlow protocol, where a set of defined messages can be exchanged between these entities over a secure channel [8]. The secure channel is the interface that connects each OpenFlow switch to a controller. The Transport Layer Security (TLS) connection to the user-defined (otherwise fixed) controller is initiated by the switch on its power on. The controller's default TCP port is 6633. The switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key. Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate).

The OpenFlow protocol supports three message types, controller-to-switch, asynchronous, and symmetric, each with multiple sub-types[9]. Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the state of the switch. Asynchronous messages are initiated by the Switch and used to update the controller of network events and changes to the switch state. Symmetric Messages are initiated by either the switch or the controller and sent without solicitation. The message types used by OpenFlow are described below. Can we see this message by wireshark program in Fig. 3.

Message between controller and switches by wireshark program Fig 3.

➢**Symmetric**

Symmetric messages are sent without solicitation, in either direction.

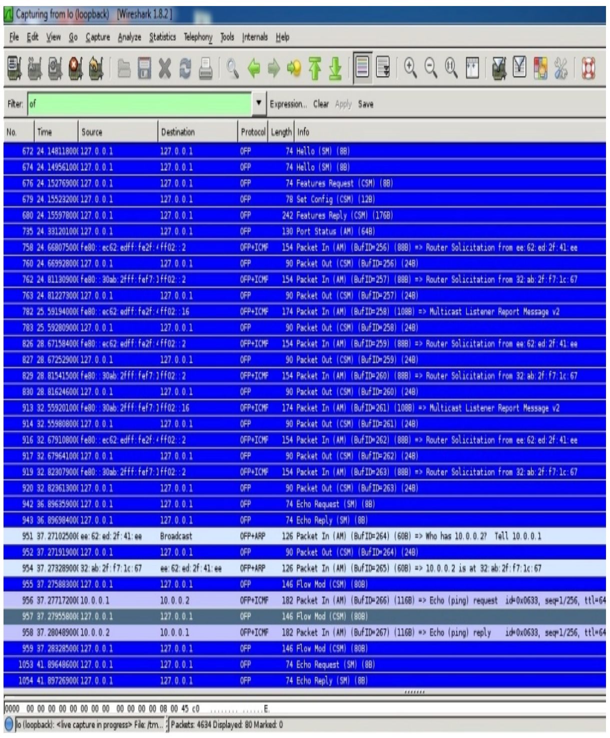
Hello: Hello messages are exchanged between the switch and controller upon connection startup.

Echo: Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They are mainly used to verify the liveness of a controller-switch connection, and may as well be used to measure its latency or bandwidth.

Experimenter: Experimenter messages provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

➢**Controller-to-Switch**

Features: The controller may request the capabilities of a switch by sending a features request


Capturing from lo (loopback) [Wireshark 1.8.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save

```
No.   Time        Source            Destination        Protocol  Length Info
  672 24.14811800( 127.0.0.1         127.0.0.1          OFP        74 Hello (SM) (8B)
  674 24.14956100( 127.0.0.1         127.0.0.1          OFP        74 Hello (SM) (8B)
  676 24.15276900( 127.0.0.1         127.0.0.1          OFP        74 Features Request (CSM) (8B)
  679 24.15523200( 127.0.0.1         127.0.0.1          OFP        78 Set Config (CSM) (12B)
  680 24.15597800( 127.0.0.1         127.0.0.1          OFP       242 Features Reply (CSM) (176B)
  735 24.33120100( 127.0.0.1         127.0.0.1          OFP       130 Port Status (AM) (64B)
  758 24.66807500( fe80::ec62:edff:fe2f:4 ff02::2       OFP+ICMP  154 Packet In (AM) (BufID=256) (88B) => Router Solicitation from ee:62:ed:2f:41:ee
  760 24.66992800( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=256) (24B)
  762 24.81130900( fe80::30ab:2fff:fef7:1 ff02::2       OFP+ICMP  154 Packet In (AM) (BufID=257) (88B) => Router Solicitation from 32:ab:2f:f7:1c:67
  763 24.81227300( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=257) (24B)
  782 25.59194000( fe80::ec62:edff:fe2f:4 ff02::16      OFP+ICMP  174 Packet In (AM) (BufID=258) (108B) => Multicast Listener Report Message v2
  783 25.59280900( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=258) (24B)
  826 28.67158400( fe80::ec62:edff:fe2f:4 ff02::2       OFP+ICMP  154 Packet In (AM) (BufID=259) (88B) => Router Solicitation from ee:62:ed:2f:41:ee
  827 28.67252900( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=259) (24B)
  829 28.81541500( fe80::30ab:2fff:fef7:1 ff02::2       OFP+ICMP  154 Packet In (AM) (BufID=260) (88B) => Router Solicitation from 32:ab:2f:f7:1c:67
  830 28.81624600( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=260) (24B)
  913 32.55920100( fe80::30ab:2fff:fef7:1 ff02::16      OFP+ICMP  174 Packet In (AM) (BufID=261) (108B) => Multicast Listener Report Message v2
  914 32.55980800( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=261) (24B)
  916 32.67910800( fe80::ec62:edff:fe2f:4 ff02::2       OFP+ICMP  154 Packet In (AM) (BufID=262) (88B) => Router Solicitation from ee:62:ed:2f:41:ee
  917 32.67964100( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=262) (24B)
  919 32.82307900( fe80::30ab:2fff:fef7:1 ff02::2       OFP+ICMP  154 Packet In (AM) (BufID=263) (88B) => Router Solicitation from 32:ab:2f:f7:1c:67
  920 32.82361300( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=263) (24B)
  942 36.89635900( 127.0.0.1         127.0.0.1          OFP        74 Echo Request (SM) (8B)
  943 36.89698400( 127.0.0.1         127.0.0.1          OFP        74 Echo Reply (SM) (8B)
  951 37.27102500( ee:62:ed:2f:41:ee Broadcast          OFP+ARP   126 Packet In (AM) (BufID=264) (60B) => Who has 10.0.0.2?  Tell 10.0.0.1
  952 37.27191900( 127.0.0.1         127.0.0.1          OFP        90 Packet Out (CSM) (BufID=264) (24B)
  954 37.27328900( 32:ab:2f:f7:1c:67 ee:62:ed:2f:41:ee  OFP+ARP   126 Packet In (AM) (BufID=265) (60B) => 10.0.0.2 is at 32:ab:2f:f7:1c:67
  955 37.27588300( 127.0.0.1         127.0.0.1          OFP       146 Flow Mod (CSM) (80B)
  956 37.27717200( 10.0.0.1          10.0.0.2           OFP+ICMP  182 Packet In (AM) (BufID=266) (116B) => Echo (ping) request  id=0x0633, seq=1/256, ttl=64
  957 37.27955800( 127.0.0.1         127.0.0.1          OFP       146 Flow Mod (CSM) (80B)
  958 37.28048900( 10.0.0.2          10.0.0.1           OFP+ICMP  182 Packet In (AM) (BufID=267) (116B) => Echo (ping) reply    id=0x0633, seq=1/256, ttl=64
  959 37.28328500( 127.0.0.1         127.0.0.1          OFP       146 Flow Mod (CSM) (80B)
 1053 41.89648600( 127.0.0.1         127.0.0.1          OFP        74 Echo Request (SM) (8B)
 1054 41.89726900( 127.0.0.1         127.0.0.1          OFP        74 Echo Reply (SM) (8B)

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0   ........ ......E.

lo (loopback): <live capture in progress> File: /tm...   Packets: 4634 Displayed: 80 Marked: 0
```

Fig. 3. Controller/switch messages are initiated by the controller and may or may not require a response from the switch

Configuration: The controller is able to set and query configuration parameters in the switch.

Modify-State: Modify-State messages are sent by the controller to manage state on switches.

Read-State: Read-State messages are used by the controller to collect various information from the switch, such as current Configuration, statistics and capabilities.

Packet-out: These are used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages.

Barrier: Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

Role-Request: Role-Request messages are used by the controller to set the role of its OpenFlow channel, or query that role.

Asynchronous-configuration: The Asynchronous-configuration message are used by the controller to set an additional filter on the asynchronous messages that it wants to receive on its OpenFlow channel, or to query that filter.

- **Asynchronous**

Asynchronous messages are sent without a controller soliciting them from a switch. Switches send asynchronous messages to controllers to denote a packet arrival, switch state change, or error. The four main asynchronous message types are described below.

Packet-in: Transfer the control of a packet to the controller. For all packets forwarded to the controller reserved port using a flow entry or the table-miss flow entry, a packet-in event is always sent to controllers.

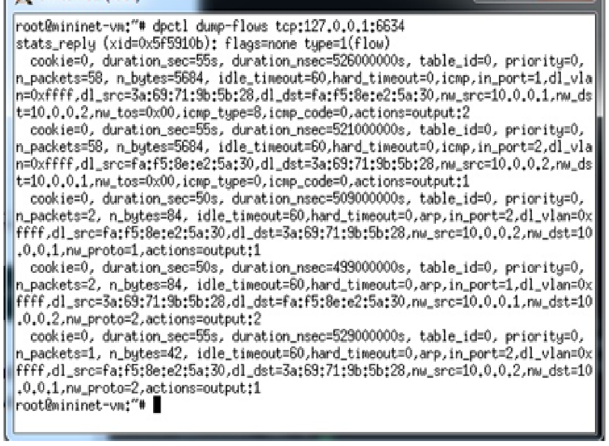
Flow-Removed: Inform the controller about the removal of a flow entry from a flow table

Port-status: Inform the controller of a change on a port.

Error: The switch is able to notify controllers of problems using error messages.

3. 3. Controller Adds Flow Entries In Switch

The Openflow switch, consists of a **flow table** containing flow entries, used to perform packet lookup and forwarding and a **secure channel** to the controller, through which Openflow messages are exchanged between the switch and the controller [10]. Can we see flow entries by by Mininet in Fig.4.



```
root@mininet-vm:~# dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x5f5910b): flags=none type=1(flow)
  cookie=0, duration_sec=55s, duration_nsec=526000000s, table_id=0, priority=0,
n_packets=58, n_bytes=5684, idle_timeout=60,hard_timeout=0,icmp,in_port=1,dl_vla
n=0xffff,dl_src=3a:69:71:9b:5b:28,dl_dst=fa:f5:8e:e2:5a:30,nw_src=10.0.0.1,nw_ds
t=10.0.0.2,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=output:2
  cookie=0, duration_sec=55s, duration_nsec=521000000s, table_id=0, priority=0,
n_packets=58, n_bytes=5684, idle_timeout=60,hard_timeout=0,icmp,in_port=2,dl_vla
n=0xffff,dl_src=fa:f5:8e:e2:5a:30,dl_dst=3a:69:71:9b:5b:28,nw_src=10.0.0.2,nw_ds
t=10.0.0.1,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=output:1
  cookie=0, duration_sec=50s, duration_nsec=509000000s, table_id=0, priority=0,
n_packets=2, n_bytes=84, idle_timeout=60,hard_timeout=0,arp,in_port=2,dl_vlan=0x
ffff,dl_src=fa:f5:8e:e2:5a:30,dl_dst=3a:69:71:9b:5b:28,nw_src=10.0.0.2,nw_dst=10
.0.0.1,nw_proto=1,actions=output:1
  cookie=0, duration_sec=50s, duration_nsec=499000000s, table_id=0, priority=0,
n_packets=2, n_bytes=84, idle_timeout=60,hard_timeout=0,arp,in_port=1,dl_vlan=0x
ffff,dl_src=3a:69:71:9b:5b:28,dl_dst=fa:f5:8e:e2:5a:30,nw_src=10.0.0.1,nw_dst=10
.0.0.2,nw_proto=2,actions=output:2
  cookie=0, duration_sec=55s, duration_nsec=529000000s, table_id=0, priority=0,
n_packets=1, n_bytes=42, idle_timeout=60,hard_timeout=0,arp,in_port=2,dl_vlan=0x
ffff,dl_src=fa:f5:8e:e2:5a:30,dl_dst=3a:69:71:9b:5b:28,nw_src=10.0.0.2,nw_dst=10
.0.0.1,nw_proto=2,actions=output:1
root@mininet-vm:~# 
```

Fig. 4. Flow entries

Main components of a flow entry in a flow table [11].

- **Match fields**: to match against packets. These consist of the ingress port and packet headers, and optionally metadata specied by a previous table.
- **Priority**: matching precedence of the flow entry.
- Counters: updated when packets are matched.
- **Instructions**: to modify the action set or pipeline processing.
- **Timeouts**: maximum amount of time or idle time before flow is expired by the switch.
- **Cookie**: opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion. Not used when processing packets.

4. Design topologies in Mininet

The SDN emulator needs topologies that are defined in Python for its execution, since all topologies available quite similar structure, parsing them to generate executable Mininet topologies is possible with ease. Topologies to be used in Mininet are executable/loadable Python classes interfacing with the Mininet API. So each usable Mininet topology is similar, having the same content in the head and the tail of a file. The difference between executable and loadable files is in the code at the end of the Python script. If the topology is executed from a Linux shell, Mininet is automatically started with the topology defined and secure shell (SSH) access available. Loaded means that Mininet was started alone from the Linux shell and the topology was given as a calling argument via the topo parameter. The code that defines SSH access to the topology nodes is not executed, only the part defining that the topology is used. In return this means using the topology just by loading SSH access is not available. So the preferred usage is to directly execute the topology from a Linux shell.

Mininet can use. An adjustment of the bandwidth in M bps is also possible by setting the B.w value of all edges to the given value. Further, the parser requires the files to be located in the same directory and without specifying input parameters the program will terminate. However, some values can be omitted, like the bandwidth limitation, which is otherwise initialized to 10M bps. If omitted, the remote controller IP is initialized with”10.0.2.2”, which is the standard IP for the host OS when using Oracle Virtualbox for virtualization.

4. 1. Opendaylight controller

OpenDaylight is an open source project with a modular, pluggable, and flexible controller platform at its core. This OpenFlow controller is implemented strictly in software and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

OpenDaylight is a community to promote and/or propose standardization of SDN northbound APIs, so that services that use an open flow controller can be written quickly and effectively. This controller is based on OSGi (Open Serices Gateway initiative) framework and it exposes REST (Representational State Transfer - a web based) API.

The controller platform itself contains a collection of dynamically pluggable modules to perform needed network tasks. There are a series of base network services for such tasks as understanding what devices are contained within the network and the capabilities of each, statistics gathering, etc. In addition, platform oriented services and other extensions can also be inserted into the controller platform for enhanced SDN functionality.

The southbound interface is capable of supporting multiple protocols (as separate plugins), e. g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. These modules are dynamically linked into a Service Abstraction Layer (SAL). The SAL exposes device services to which the modules north of it are written. The SAL determines how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices [12].

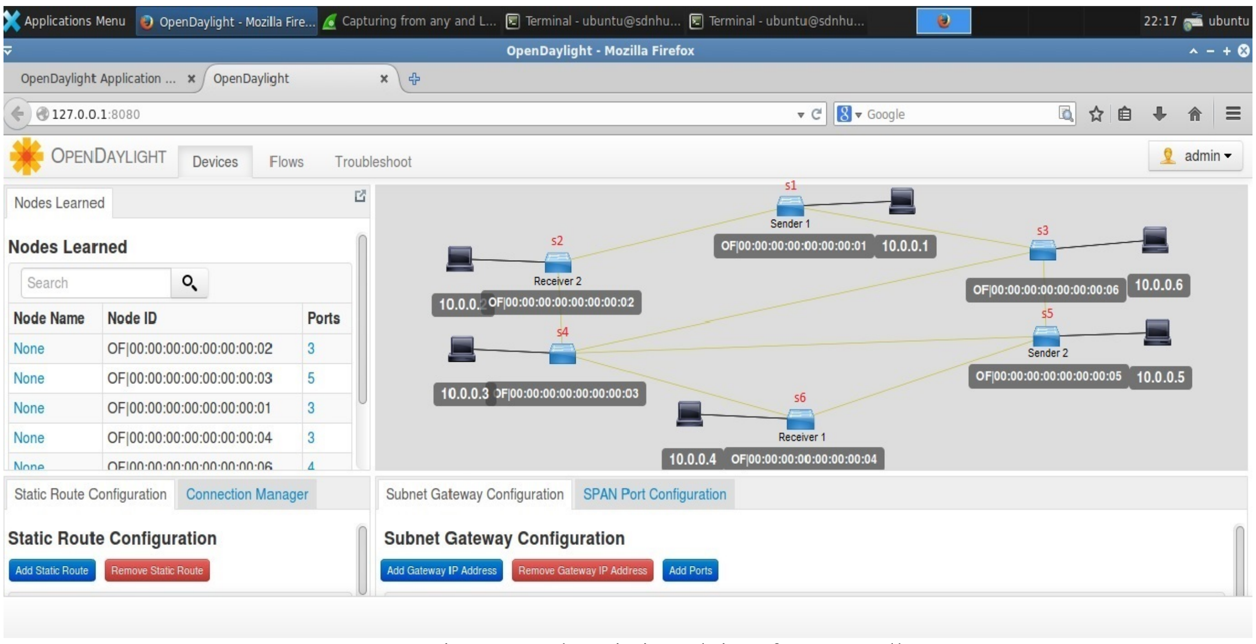
4. 2. Internet Traffic Generator (D-ITG)

To evaluate the performance of Mininet D-ITG in version 2.8.0-rc1 was used: “Distributed Internet Traffic Generator (D-ITG) is a platform capable to produce traffic that accurately adheres to patterns defined by the inter departure time between packets (IDT) and the packet size (PS) stochastic processes” [13]. Therefore, it offers a rich variety of probability distributions for the traffic generation and uses some models proposed to emulate sources of various protocols. With it, it is possible to generate various packet streams and collect statistics with a logging server. Important modules of the D-ITG are depicted. The ITGSend module is responsible for the traffic generation, while the ITGRecv module is the sink for the packets, which are delivered over a Data Channel. To collect logging information both, the ITGSend and ITGRecv are communicating via a Log Channel with the ITGLog module. For remote control the ITGManager offers the functionalities to adjust parameters of ITGSend through the Signaling Channel.

4. 3. Measurement Trials

Fig. 5 depicts the topology of our measurement trials. S1 to S6 are the switches in the topology, while Sender/Receiver denote the hosts that are handling generated traffic. On each sender a ITGSend process is called to generate the traffic, while on each receiver ITGRecv handles the receipt of the packets. Besides, the log server collects the relevant statistical data of the hosts by running an instance of ITGLog. S4 is shutdown for a few measurement trials and therefore all dashed links are unavailable. To sum up, in total we performed trials, each for TCP and UDP with a duration of one minute for each trial.

The D-ITG decoder provides data files that can be analyzed with MatLab. Initially, every trial is evaluated and plotted as shown in Fig. 6. In the plots, graphs show specific characteristics in the time sequence from 0 s to 60 s. In more detail, the upper left plot depicts the throughput in M bps, while the upper right shows the delay in ms. The lower left plot is evaluating the jitter value in ms.


Applications Menu
OpenDaylight - Mozilla Fire...
Capturing from any and L...
Terminal - ubuntu@sdnhu...
Terminal - ubuntu@sdnhu...
22:17
ubuntu
OpenDaylight - Mozilla Firefox
OpenDaylight Application ...
OpenDaylight
127.0.0.1:8080
Google
OPENDAYLIGHT
Devices
Flows
Troubleshoot
admin
Nodes Learned
s1
Sender 1
OF|00:00:00:00:00:00:00:01
10.0.0.1
s2
Receiver 2
10.0.0.2
OF|00:00:00:00:00:00:00:02
s3
OF|00:00:00:00:00:00:00:06
10.0.0.6
s4
s5
Sender 2
OF|00:00:00:00:00:00:00:05
10.0.0.5
10.0.0.3
OF|00:00:00:00:00:00:00:03
s6
Receiver 1
10.0.0.4
OF|00:00:00:00:00:00:00:04
Static Route Configuration
Connection Manager
Add Static Route
Remove Static Route
Subnet Gateway Configuration
SPAN Port Configuration
Add Gateway IP Address
Remove Gateway IP Address
Add Ports

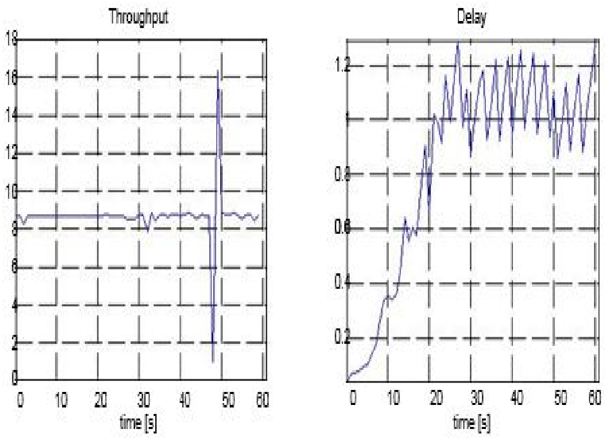
Nodes Learned

Node Name	Node ID	Ports
None	OF|00:00:00:00:00:00:00:02	3
None	OF|00:00:00:00:00:00:00:03	5
None	OF|00:00:00:00:00:00:00:01	3
None	OF|00:00:00:00:00:00:00:04	3
None	OF|00:00:00:00:00:00:00:06	4

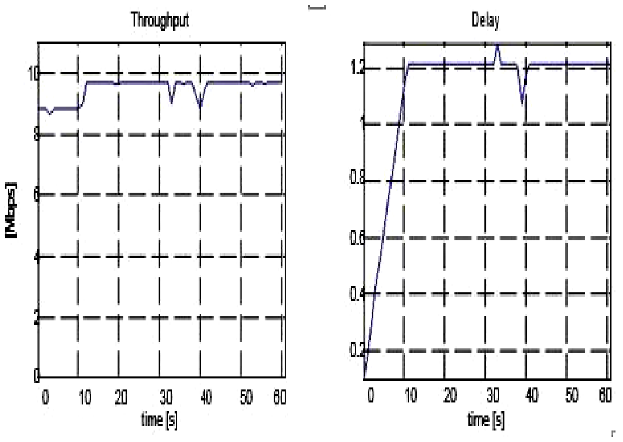
Static Route Configuration

Subnet Gateway Configuration

Fig. 5. Topology in in web interface controller



a



b

Fig. 6. Evaluation of traffic with a constant test bitrate of 10Mbps. The delay within the edges of the SDN topology is 1ms: *a*–Traffic generation over TCP *b* –Traffic generation over UDP

5. Conclusion & future work

This paper presents SDN technology We have analyzed all messages between switch open flow and controller and captured in wireshark, Allows us to understand the relationship between controller and switch openflow and relationship between switches.

In second part we have used the simulators mininet and built by the new topology Consists of 6 switches and 6 host in the Python languages, all this devices controlled by opendaylight controller and D-ITG program for Generate traffic between devices we show the results in graphs.

References

1. Khatri, V. (2013). Khatri vikramajeet analysis of openflow protocol in local area net- degree programme in information technology. Tampere university of technology, 74.

2. Hegr, T., Bohac, L., Uhlir, V., Chlumsky, P. (2013). OpenFlow Deployment and Concept Analysis. Information and communication technologies and services, 11 (5), 327–335.

3. Open New York Foundation Software-Defined Networking (2012). The New Norm for Networks.

4. Octopress, M. T.-P. (2013). A Quarterly Technical Publication for Internet and Intranet Professionals. A Quarterly Technical Publication for Internet and Intranet Professionals.

5. Open Networking Foundation (2013). OpenFlow Switch Specification, 1–205.

6. Mininet Team-Powered by Octopress (2014). Mininet Overview. Available at: http://mininet.org/overview/

7. Openflow (2011). Explain all the requirements to run Mininet.

8. Azodolmolky, S. (2013). Software Defined Networking with OpenFlow. Packt Publishing, 153.

9. Open Networking Foundation (2014). OpenFlow Switch Specification, 4, 1–171.

10. Kontesidou, G., Zarifis, K. (2009). Openflow Virtual Networking : A Flow- Based Network Virtualization Architecture Openflow Virtual Networking : A Flow-Based. Royal Institute of Technology.

11. Technical Solution Guide (2013). HP OpenFlow Protocol Overview, 18.

12. Opendaylight. Available: http://www.opendaylight.org/ project/tech nical-overview

13. S. Avallone, S., Guadagno, S., Emma, D. (2004). D-ITG Distributed Internet Traffic Generator. University's di Napoli Federico II COMICS Lab, Department di Informatics e Sistemistica, 4.

Рекомендовано до публікації д-р техн. наук Тіхонов В. І.
Дата надходження рукопису 25.09.2014

Taher Abdullah, PHD, Telecommunication systems department, Odessa National Academy of Telecommunications named after O. S. Popov, Ukraine
E-mail: abidalla_2004@yahoo.com