Developing a Multipath-TCP Analyzer using Software Defined Networking

Carlos Cajas Eng.1, Christian Valdivieso Eng.[1], David Mejía M.Sc.[1], and Iván Bernal Ph.D. [1]
[1]Escuela Politécnica Nacional, Ecuador, {carlos.cajas; christian.valdivieso; david.mejia; ivan.bernal}@epn.edu.ec

Abstract– MP-TCP (MultiPath-Transmission Control Protocol) is a network protocol that uses subflows for allowing the existence of disjoint paths and increases the overall throughput with respect to employing a common TCP connection. The idea of analyzing MP-TCP messages using the principles of SDN (Software Defined Networking) is proposed. Relevant aspects of how the Analyzer was developed as a module for OpenDayLight's SDN framework are presented. The Analyzer runs in the SDN controller and it commands the installation of appropriate rules in the network devices (switches supporting Openflow) so that all TCP traffic be derived to the controller. Then the Analyzer must identify MP-TCP messages by checking the options field of TCP and present all the related information to the user employing a GUI (Graphical User Interface). By combining the usage of proactive and reactive rules, the Analyzer's implementation tries to minimize the impact of sending TCP traffic to the controller. The Analyzer has been tested with other modules (e.g. Layer 2 Switch) that are included with OpenDayLight; this is done for showing that it is possible that the Analyzer may not interfere with other running modules. Some results obtained with the Analyzer when using physical and virtual switches in different topologies in a network using MP-TCP are presented. Simulation results using MP-TCP with Mininet are also discussed.

Keywords—Protocol Analyzer, MP-TCP, SDN, Openflow, OpenDayLight.

Digital Object Identifier (DOI):
http://dx.doi.org/10.18687/LACCEI2017.1.1.264
ISBN: 978-0-9993443-0-9
ISSN: 2414-6390

Developing a Multipath-TCP Analyzer using Software Defined Networking

Carlos Cajas Eng.[1], Christian Valdivieso Eng.[1], David Mejía M.Sc.[1], and Iván Bernal Ph.D. [1]
[1]Escuela Politécnica Nacional, Ecuador, *{carlos.cajas; christian.valdivieso; david.mejia; ivan.bernal}@epn.edu.ec*

Abstract– MP-TCP (MultiPath-Transmission Control Protocol) is a network protocol that uses subflows for allowing the existence of disjoint paths and increases the overall throughput with respect to employing a common TCP connection. The idea of analyzing MP-TCP messages using the principles of SDN (Software Defined Networking) is proposed. Relevant aspects of how the Analyzer was developed as a module for OpenDayLight's SDN framework are presented. The Analyzer runs in the SDN controller and it commands the installation of appropriate rules in the network devices (switches supporting Openflow) so that all TCP traffic be derived to the controller. Then the Analyzer must identify MP-TCP messages by checking the options field of TCP and present all the related information to the user employing a GUI (Graphical User Interface). By combining the usage of proactive and reactive rules, the Analyzer's implementation tries to minimize the impact of sending TCP traffic to the controller. The Analyzer has been tested with other modules (e.g. Layer 2 Switch) that are included with OpenDayLight; this is done for showing that it is possible that the Analyzer may not interfere with other running modules. Some results obtained with the Analyzer when using physical and virtual switches in different topologies in a network using MP-TCP are presented. Simulation results using MP-TCP with Mininet are also discussed.

Keywords—Protocol Analyzer, MP-TCP, SDN, Openflow, OpenDayLight.

I. INTRODUCTION

Nowadays, many devices such as laptops, tablets and smartphones are multihomed using different interfaces to connect to a network (Wi-Fi or a mobile network). However, TCP (Transmission Control Protocol) does not take advantage of multiple interfaces in hosts. Even though there may exist multiple paths in a network, TCP sessions are limited to take advantage of only one path. A related problem is due to moving hosts that may require changing the current network configuration, a process that demands resetting IP (Internet Protocol) and TCP values.

MP-TCP (MultiPath TCP) is a network protocol that can be considered a TCP extension for multipath forwarding. MP-TCP is an alternative to exploit multihomed scenarios. A MP-TCP connection splits into several TCP connections referred to as subflows. These subflows can use disjoint paths using either a single interface or multiple interfaces; this may allow increasing the throughput of the overall connection. Besides, even if a subflow fails during the live time span of a MP-TCP connection, the connection may continue by using the remaining subflows. This strategy increases reliability if something goes wrong in any of the network paths [1].

On the other hand, SDN (Software Defined Networking) technology has emerged as a flexible way to control the network in a systematic way. Networks have as two of their building blocks a data plane and a control plane which coexist in a unique physical device in traditional networking architectures. In SDN the control plane is physically separated and moved to a computer known as a controller. The data plane is implemented by networking devices that communicate with a controller by means of some protocol for receiving instructions of how to handle incoming traffic. By installing rules in the switches, a controller has a high level of control of the network which in turn makes introducing changes, new functionalities and services faster and easier.

Based on this flexibility, the idea proposed in this paper is that monitoring specific protocols in networking devices along the path that traffic flows follow can benefit from SDN. This technique is applied for monitoring and analyzing the messages of MP-TCP by using Openflow devices and OpenDayLight's platform [2]. A module named as "MP-TCP Analyzer" is developed to run in OpenDayLight's context. The Analyzer obtains TCP packets from the switching devices before they get to the destination hosts, then it decodes MP-TCP information and presents the results to the network administrator in a GUI (Graphical User Interface).

Non-SDN solutions for monitoring and analyzing protocols include port mirroring that allows to copy traffic of a live network and send it to port where an analysis tool is connected, using extra ports on a switch [3]. With port mirroring it is not possible to select traffic, and the switch can be overloaded with the volume of traffic being mirrored.

In response to these issues, the industry came up with dedicated devices called network packet brokers [4] that can select the traffic being monitored (based on IP address or application type, for example) and forward it to analysis tools; however, they are too expensive to use everywhere one might want to monitor or analyze traffic. Another option is to use network Test Access Points (TAPs) [5] which are hardware tools that allow to access and monitor a network.

SDN opens the possibility of implementing more inexpensive alternatives with low-cost switches and controllers and can offer granular traffic control comparable to network packet brokers. So SDN is a feasible way to develop a simple and dynamic packet monitoring system to analyze the traffic of a network.

Within the SDN alternative, one way is not replacing monitoring tools but just changing the way in which

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2017.1.1.264
ISBN: 978-0-9993443-0-9
ISSN: 2414-6390

information is passed from the physical network to analyzing tools. This can be implemented using the same idea of port mirroring, in this case controlling the rules in the switches but this maintains the aforementioned problem. A tool using this idea is mentioned in [6]. Another way is to develop tools that simply collect information gathered in the controllers [7]. Besides, analyzers using the SDN architecture usually lack the packet decode for protocol details required for some analysis and troubleshooting.

The Analyzer that is proposed in this paper is based on SDN, it presents details that go to bit level, it is run in the context of the controller and it uses the same port that connects switches with the controller. It does have some shortcomings that will be mentioned in later sections.

Another aspect to consider is that the development of the MP-TCP analyzer was proposed as part of a larger project that implements a solution for the shared bottleneck [8] that MP-TCP suffers. Details at bit level for MP-TCP were not offered by other available tools and these details were required for checking for compliance and reproducing the shared bottleneck.

The remaining of the paper is organized as follows: Section II outlines some useful details about MP-TCP. This work will not dive deep into MP-TCP's specification, but some basic technical details should be understood and are briefly discussed. Section III presents some aspects regarding SDN, including Openflow and OpenDayLight. Section IV outlines some aspects related to the design and implementation of the Analyzer, including the GUI. Section V presents some results obtained from running tests with the Analyzer and using different network topologies. Finally, conclusions are outlined in Section VI.

II. MP-TCP

MP-TCP is a modification or extension to TCP. MP-TCP is specified in the *Request For Comment* RFC-6824 of the Internet Engineering Task Force (IETF) [9].

Each subflow in MP-TCP is characterized by a different tuple of elements which are: (source IP address, destination IP address, source port, destination port, protocol identifier). Every subflow creates a TCP connection, the set of subflows form the overall MP-TCP connection.

By using multiple paths and by sending data through the less congested paths [1], both throughput and redundancy are increased and the grade of use of the available resources is maximized. It is also possible to perform load balancing between available paths.

MP-TCP is transparent for the higher and lower layers of the TCP / IP architecture. This allows user-level applications to simply use MP-TCP, assuming the kernel of the operating system supports MP-TCP.

A. *MP-TCP Option Subtypes*

It is worth mentioning that it is not accurate to introduce the idea of "MP-TCP packets", packets are simply standard compliant TCP packets using their option field to handle MP-TCP data. Sometimes the term "MP-TCP messages" is used for referring to TCP's option field when it contains MP-TCP information. MP-TCP option format (Fig. 1) includes the following fields:

1) Kind: 8-bit field with a value of 30. This field allows to discriminate between MP-TCP and other TCP options.

2) Length: 8-bit field for signaling the size of the MP-TCP message.

3) Subtype: 4-bit field for discriminating among MP-TCP messages (see Table I).

4) Subtype-specific data: Space where each MP-TCP subtype messages place their control information.

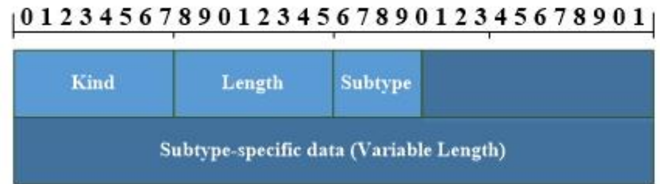

Fig. 1 MP-TCP option format.

Some examples of the tasks associated to MP-TCP that use several of the option subtypes are:

1) Establish a MP-TCP connection: An MP_CAPABLE message declares MP-TCP support during the connection setup. This message is part of TCP's three-way handshake. The first connection will be the main subflow of the MP-TCP connection.

2) Add new subflows: MP-TCP uses ADD_ADDRESS messages for notifying the hosts about a new IP address. This address can be used to establish a new subflow. After an ADD_ADDRESS message is received by the hosts, they may send MP_JOIN messages.

3) Disable offline paths: When an interface goes down or a subflow connection cannot be supported any more (e.g. a middlebox is present somewhere along the path), a REMOVE_ADDRESS message is sent. This message allows disabling subflows related to this address.

TABLE I
MP-TCP SUBTYPE FIELD

Value	Symbol	Name
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DSS	Data Sequence Signal
0x3	ADD_ADDRESS	Add Address
0x4	REMOVE_ADDRESS	Remove Address
0x5	MP_PRIO	Change Subflow Priority
0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast Close

4) *Signaling MP-TCP data*: With a Data Sequence Signal (DSS) message, MP-TCP connections can signal MP-TCP data. There is a data sequence number (DSN) and there is a subflow sequence number. Data is distributed among subflows so data sequence mapping is required. A DSS message helps to fit each subflow into the overall MP-TCP stream.

5) *Establish priority*: An MP_PRIO message can change the state of a subflow from backup to regular. If the subflow is established like a backup path, it will not be use in a MP-TCP connection unless all available subflows are down.

6) *Close connection*: When a whole MP-TCP connection abruptly crashes, MP_FASTCLOSE messages are sent. This message is similar to RST in regular TCP connections.

7) *Fallback to a regular TCP connection*: In some cases, middleboxes might be present and might not support MP-TCP. These devices could erase the whole TCP option field or make payload data be lost. In these special scenarios, it is necessary to fallback to regular TCP connections; this is achieved with an MP_FAIL message.

B. *MP_CAPABLE Message*

The Analyzer must decode MP-TCP messages by discriminating which of the subtype messages is contained and decode all the fields that the specific message may have. For this reason, it is necessary to know all the fields of every MP-TCP subtype messages. As an example, the structure of MP_CAPABLE messages (see Fig. 2) is presented, followed by a short description of each field.

Fig. 2 MP_CAPABLE option.

1) *Version*: Signals MP-TCP version. 0 for every current available implementation.

2) *Bit A*: A value of 1 indicates that a checksum is required. It is not required that both sender and receiver set their bit A to 1. Only if both hosts set their bit A to 0, a checksum will not be used.

3) *Bit B*: It must be set to 0. This bit will be used in future implementations.

4) *Bits C-H*: These 6 bits allow to negotiate the crypto algorithm to be used. Currently only bit H is used and set to 1. The other bits C to G are set to 0. With this configuration, MP-TCP will use HMAC-SHA1 as the crypto algorithm. The algorithm and the keys will be used in MP_JOIN messages for authenticating the connection and avoid authentication attacks.

5) *Sender's Key*: 64-bit key generated by the sender. This key will be used with MP_JOIN messages. Besides, it is sent in SYN and ACK packets.

6) *Receiver's Key*: 64-bit key generated by the receiver. It will be used by MP_JOIN messages. This key is sent in SYN/ACK and ACK packets. If the receiver gets both keys correctly in ACK packets, the MP-TCP connection is established.

Additional information about every MP-TCP message is available in [1].

III. SDN (SOFTWARE DEFINED NETWORKING)

As mentioned before, SDN is a networking architecture that decouples the control and switching planes [10]. The switches or white boxes are required to be extremely efficient at their task of switching and must reduce their intelligence to a minimum.

The intelligence of the control plane is derived to a controller (Fig. 3) that executes software modules that define the functionality of the switches and generate rules that must be installed on them. The controller communicates with the switches by means of a protocol (e.g. Openflow); in the switch side this protocol allows manipulating the flow table of the switch.

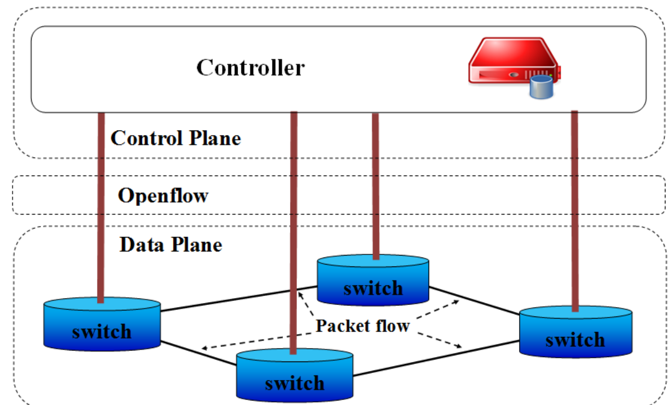

Fig. 3 SDN architecture.

These changes in the architecture compared to traditional networks will enable the development of a highly flexible infrastructure that will allow operators the deployment and introduction of new services into the network at an increased pace of innovation.

Some of the key characteristics of SDN are:

- Directly program network devices from the controller, thanks to the separation of the control and data planes.
- Dynamically manage the network, adjusting to the traffic that circulates in it, managing the eventual requirements that may appear.

- Centrally control and manage the functionality of the network. This is because the controllers can handle a set of devices and can decide the entire operation of the network applying previously defined policies.
- Configure the use of network resources dynamically. Network operators can write their programs achieving a rapid innovation since there would not exist dependency on proprietary software.
- Simplify the design of the network as well as its operation, since the instructions are received from the controller, instead of being received from vendors specific to certain brands or protocol owners.

A. *Openflow*

Openflow, created by the ONF (Open Networking Foundation), is one way of structuring the communication between the controller and switching devices. Switches that support Openflow have one or more tables holding rules. These rules handle the incoming packets, discriminate among traffic flows and execute a set of actions on them [11].

Each rule includes some fields that must be matched against incoming traffic. There is a variety of conditions that can be specified to match, such as: in-port, out-port, source ipv4 address, etc. Based on the matching result, certain actions will be executed. Actions include: forwarding to other ports, sending packets to the controller, flooding, dropping packets, etc. Depending on the rules installed on networking devices by the controller, the devices will have different functionalities and services such as: firewall services, L2 Switch capabilities, NAC (Network Access Control) services, load balancers, etc.

B. *Openflow Switches*

An Openflow switch is a solution that consists of a physical or virtual switch that has one or more internal flow tables where it is determined how to process any data flow that enters the switch. Besides, it has a channel to communicate with a SDN controller [11].

C. *OpenDayLight Controller*

OpenDayLight (ODL) is a modular open-source platform for SDN, which provides centralized, programmable control and monitoring of network devices. This controller is a collaborative project organized by the Linux Foundation and aims to accelerate the adoption of SDN worldwide. The stable version that was used in the project is referred to as Beryllium.

1) *General Architecture*

OpenDayLight's architecture is made up of the Controller Platform layer and the Service Abstraction Layer (SAL) [12] as shown in Fig. 4.

2) *MD-SAL (Model Driven-SAL)*

SAL allows applications, services and modules of the controller to communicate with other components using southbound (SB) and northbound (NB) interfaces as shown in Fig. 4. SB interfaces allow communicating with network devices and NB interfaces allow communicating with Applications. SAL determines how to respond to devices regardless of what protocols in the SB interfaces are being used, so Openflow is just an option not the only one. For the project, the MD-SAL model included in Beryllium was used. MD-SAL has the following two components:

a) A data store shared by all modules which maintains the following structure:
   - *Operational Data Store:* Keeps information about the current state of the network.
   - *Configurational Data Store*: Space where modules can store configuration information. Users can also modify this store through REST APIs using NB interfaces such as is the case of DLUX.

b) A message bus is a messaging service which allows multiple modules to notify events and communicate with each other. Events includes RPCs (Remote Procedure Calls) and data change notifications.

So, SAL is the framework that connects OpenDayLight modules through well-defined APIs. Yang models are used to generate this sort of APIs [12].

Depending on whether the data is stored or read in the data store, the modules may become providers or consumers. A provider stores data in the data store and generates notifications, whereas a consumer uses the data in the data store and receives notifications about data changes.

3) *Controller Platform*

This element contains bundles or modules that implement basic functionalities or advanced networking services. L2 Switch, Topology Processing, Statics Manager and Openflow Switch Manager are examples of modules which are part of the Controller Platform [13]. The MP-TCP Analyzer is part of this layer and may interact with other modules as shown in Fig. 4.

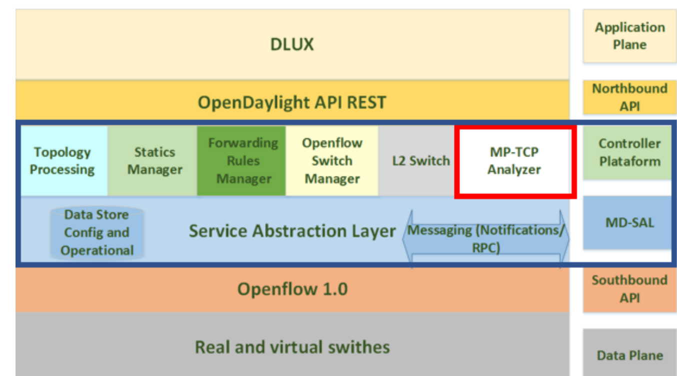

Fig. 4 MP-TCP Analyzer within OpenDayLight's architecture.

IV. Design And Implementation

A. *Functional Requirements*

Some relevant functions that the Analyzer must do are:

- Install the appropriate rules on Openflow switches (physical/real or virtual) so that they send TCP packets to the OpenDayLight controller.
- Process TCP packets received from the switches so that only packets containing their option field with MP-TCP

information are identified. Further decode MP-TCP messages of such packets.
- Present to the end user all the information obtained from MP-TCP messages using a graphical user interface.
- Display the decoded information in real time and allow the end user to filter information based on the following criteria: switch identification, MAC address, IP address, MP-TCP message subtypes.
- Start/Stop, at will, the operation of the Analyzer in either a specific switch or all the switches, without unloading the modules.

B. Use Case Diagram

Fig. 5 shows the use case diagram for the Analyzer. This diagram shows the user as the main actor who interacts with the OpenDayLight controller and MP-TCP Analyzer. The user can start/stop OpenDayLight and the Analyzer, see MP-TCP decoded messages in real time, start/stop the Analyzer functionality in every switch, filter messages based on the MP-TCP subtype, MAC addresses or IP addresses (source or destination), access help information on the Analyzer, access the graphical representation of the current topology of the network provided by DLUX.

C. Relevant components used from OpenDayLight's

The Analyzer is contained in a module that is executed in the context of OpenDayLight's platform which will be running on the SDN-controller machine. The main components from OpenDayLight's that were used in the implementation are:

DLUX: Application that exploits the REST API and is used to visualize the network topology; it is part of the modules that can be easily installed in the controller.

L2 Switch: Provides functionalities of a layer 2 switch to devices managed by the controller.

Openflow: Service located at the SB interface that manages the communication between the controller and the physical and/or virtual network devices using the Openflow protocol.

The MP-TCP Analyzer must be installed and configured to be loaded and run in Karaf [14]. It is assumed that the additional modules should have been loaded before the Analyzer.

D. Main ideas behind the implementation of the Analyzer

Java was used as the programming language for writing the code of the Analyzer. The starting point for developing the code was structured modifying an existing maven archetype [15] and Yang as the modeling language for generating classes that allow handling the data store. Then, some classes were modified and others created to develop the required functionality.

When the Analyzer starts, it registers to receive notifications for every kind of event of interest (e.g. switches connecting to the controller, packets arriving to the controller, etc.). Much of the processing must be coded when TCP packets are received in the controller.

When switches connect to the controller, the Analyzer will receive notifications associated to these events and will install proactive rules after other modules running in OpenDayLight might have processed these same notifications.

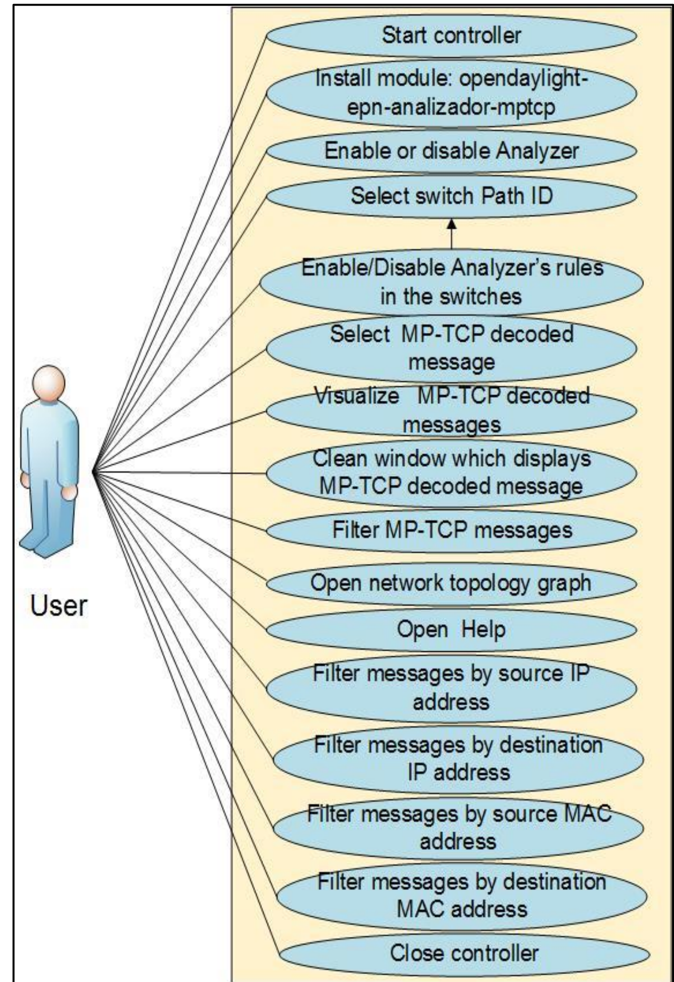

Fig. 5 Use case diagram for the MP-TCP Analyzer.

These proactive rules have higher priority than those previously installed by other modules and will have as their main action: packet flooding. Flooding will be done only in the case of incoming TCP packets and will include every Openflow forwarding port and the port connecting the switch to the controller. Flooding TCP packets may not be compatible with every peer module's functionality running in OpenDayLight.

Once the rules have been installed in the switches, every TCP packet is sent from switches to the controller. The Analyzer must discriminate between regular TCP packets and those using the option field for signaling MP-TCP. After this process, it is possible to decode each MP-TCP subtype messages.

In the case of proactive rules, depending in the network topology, the possibility of introducing loops when defining the rules should be avoided. This is accomplished by using the services of the Loop Remover Module (a submodule of L2 Switch) that allows to find out which ports are marked as forwarding or discarding. Discarding ports will not be considered in output actions such as flooding. Loop Remover applies the Spanning Tree Protocol (STP) for its tasks.

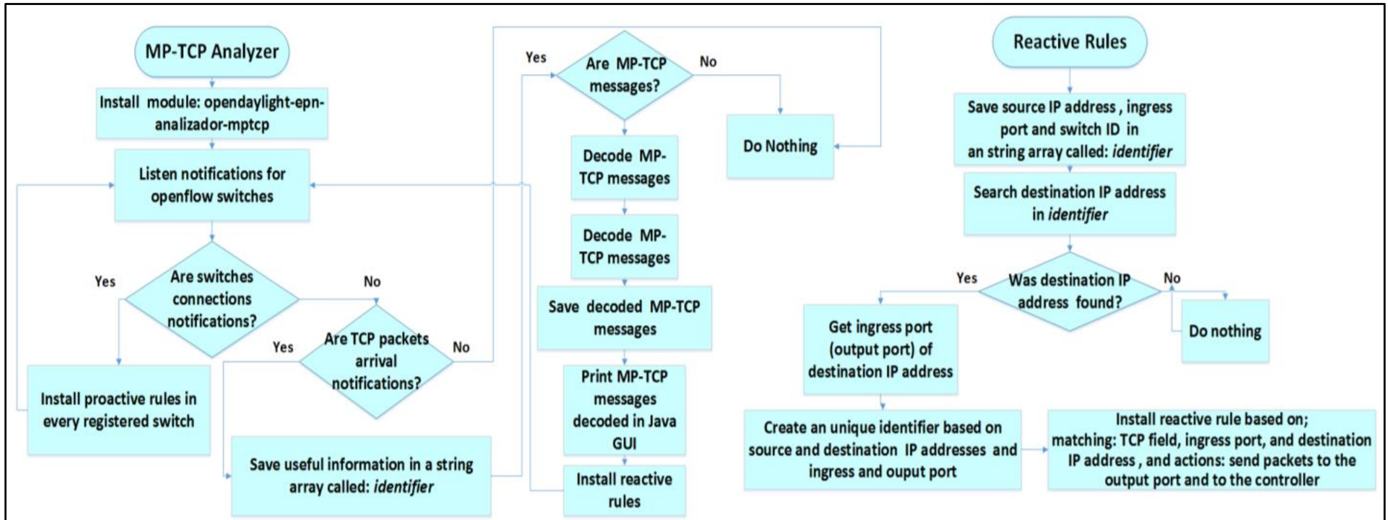

Fig. 6 Flow diagram for MP-TCP Analyzer.

For decreasing the overhead that flooding implies, as part of processing TCP traffic, reactive rules are installed on switches when initial MP-TCP messages arrive to the controller. These rules will have even higher priority than proactive rules and will avoid doing flooding when TCP packets arrive to switches but they will still be sent to the controller and be forwarded to their intended destination.

These reactive rules may also not be compatible with every peer module's functionality running in OpenDayLight, but incompatibility between modules has been reported in other scenarios [14] (e.g. Link Aggregation Control Protocol- LACP and L2 Switch).

The basic idea for the reactive rules is to find out the output port for a packet for a given destination IP address. This can be done by simply storing information contained in the arriving packets forming a key structured by: source IP address, ingress port and switch ID (identification of the switch that sent the packet to the controller). Later, when required, a search in the stored information should be made looking for the association: destination IP address, output port and switch ID. This will allow to conform the rule to be installed in a specific switch.

Finally, it should be pointed out that when TCP packets are sent to the controller, actually only 100 bytes are sent so the overhead of transmitting the full packet is avoided and this number of bytes contains the TCP option field with the MP-TCP message. 100 is a number that was estimated theoretically considering every MP-TCP message subtype and by monitoring MP-TCP implementations. TCP packets are stored in the switches for later use for forwarding them according to the installed rules.

Fig. 6 shows the flow diagram for the Analyzer which includes the different aspects that have been described in the previous paragraphs.

E. GUI for the Analyzer

A GUI for the analyzer was designed and implemented as shown in Fig.7. The upper area of the window holds a menu bar. The first menu item (File) allows visualizing previously saved MP-TCP messages. The second menu item (Actions) allows filtering MP-TCP messages based on their subtype and allows cleaning information in the decoded-packet window. The last menu item (Help) provides basic information about how to use the Analyzer.

Most of the upper section of the main window (Fig. 7) allows displaying in real time a list of the decoded MP-TCP messages which will also be persisted in files. This list for the decoded messages has been set to handle up to twenty thousand messages and will be emptied when this limit is reached. The information that is displayed for each message can be clearly observed in Fig. 7 (Path, Switch ID, etc.).

The lower left area of the main window is used to display information of the selected message in the list.

The area in the lower right section of the main window in Fig. 7 is used for three purposes:

- It is possible to input MAC and IP addresses for filtering actions. Addresses may be either source or destination.
- By using the upper comboBox, a user can select a specific switch and filter MP-TCP messages sent by the selected switch. The lower comboBox allows enabling/disabling the Analyzer in the selected switch in the upper comboBox. This is useful when the Analyzer only is required to capture MP-TCP messages coming from one switch.
- Start/Stop button helps to activate/deactivate the Analyzer. This option allows removing rules in all switches.

V. Results

For testing the Analyzer, three basic scenarios were used and they employed two HP ProCurve 3500 switches and several virtual switches based on the software implementation known as Open vSwitch (OVS) [16].

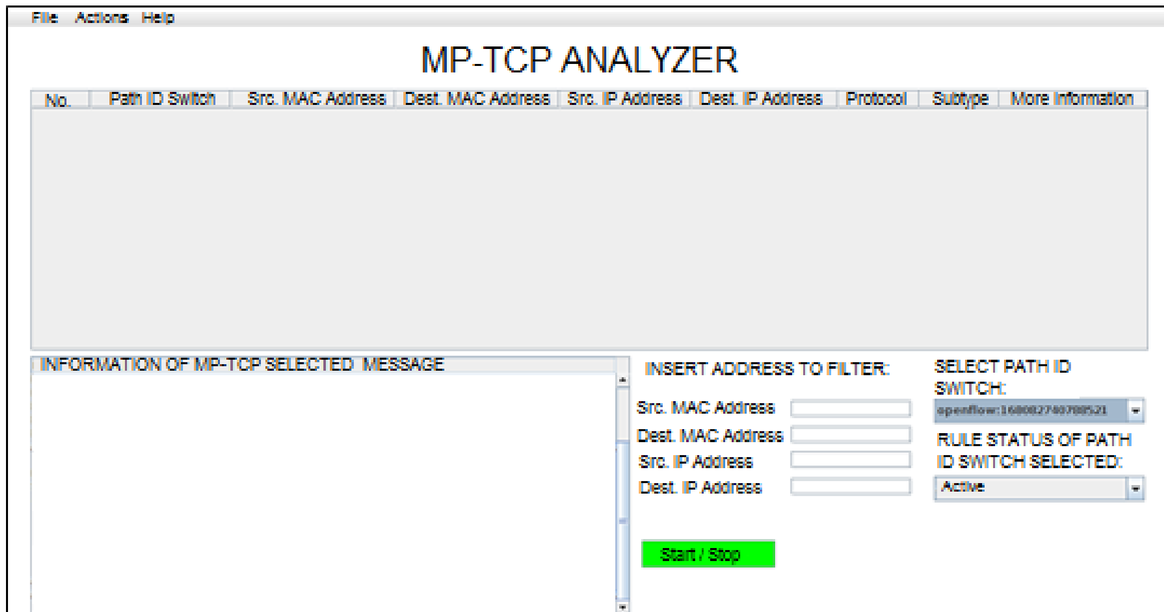

Fig. 7 MP-TCP Analyzer's main window.

A. Results for ring topology with three switches

The first topology is presented in Fig. 8. It includes two HP ProCurve 3500 switches connected to the ftp client (Switch 30) and ftp server (Switch 20). There is an alternative path that leads to a virtual switch (Switch 40) based on OVS. The client and server belong to network 192.168.50.0/24, and each of the interfaces in the controller belongs to a different network (see Fig. 8). For avoiding the controller to become overloaded, *wondershaper* was used as a tool to limit the data rate to 5Mbps in every interface.

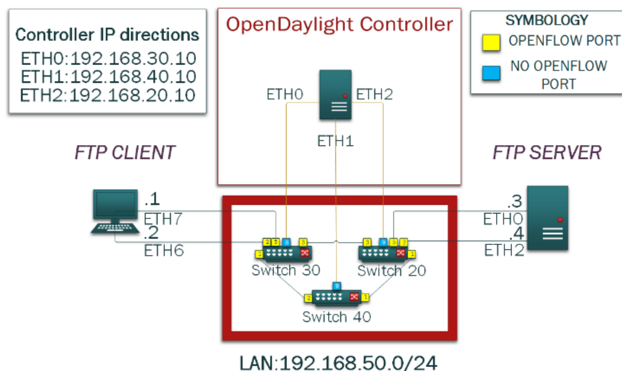

Fig. 8 First network topology for testing.

Fig. 8 shows a topology which has a loop, so the Analyzer and L2 Switch will not consider those ports labeled as *discarding* when installing rules on the switches.

Fig. 9 to 16 show some results obtained when running the MP-TCP Analyzer with the ring topology. Both L2 Switch and the Analyzer employ proactive rules. For easing up the task of distinguishing rules installed by either of these modules, it should be noted that:

- L2 Switch uses values of 0, 2 and 100 as their priority level. Rules with priority values of 200 (proactive) and 5000 (reactive) are installed by the Analyzer.
- The proactive rules include a matching condition on TCP.

Fig. 9, 11 and 13 show proactive rules installed by L2 Switch and Fig. 10, 12 and 13 show proactive rules installed by the Analyzer and are marked in green.

The proactive rules installed by L2 Switch and the Analyzer make flooding to the port that connects the switch to the controller and all Openflow ports unless a port is marked as discarding. For Switch 30 in Fig. 11, there is not any rule installed by L2 Switch whose flooding action includes port 1.

Something similar applies to Switch 40 in Fig. 13, there is not any rule associated with port 2 as an output port. This happens since both L2 Switch and the Analyzer use Loop Remover for installing their proactive flows and consider the *STP status port*.

```
controlador1@controlador1:~$ sudo ovs-ofctl dump-flows tcp:192.168.20.2:6634
NXST_FLOW reply (xid=0x4):
 cookie=0x2b0000000000000e, duration=111.164s, table=2, n_packets=2, n_bytes=
302, idle_age=71, priority=2,in_port=1 actions=output:5,output:6,output:3,ou
put:4,output:2
 cookie=0x2b0000000000000d, duration=111.164s, table=2, n_packets=0, n_bytes=
0, idle_age=111, priority=2,in_port=4 actions=output:5,output:6,output:3,outp
ut:1,output:2,CONTROLLER:65535
 cookie=0x2b0000000000000a, duration=111.165s, table=2, n_packets=0, n_bytes=
0, idle_age=111, priority=2,in_port=5 actions=output:6,output:3,output:4,outp
ut:1,output:2,CONTROLLER:65535
 cookie=0x2b0000000000000f, duration=111.164s, table=2, n_packets=2, n_bytes=
120, idle_age=9, priority=2,in_port=2 actions=output:5,output:6,output:3,outp
ut:4,output:1,CONTROLLER:65535
 cookie=0x2b0000000000000b, duration=111.165s, table=2, n_packets=0, n_bytes=
0, idle_age=111, priority=2,in_port=6 actions=output:5,output:3,output:4,outp
ut:1,output:2,CONTROLLER:65535
 cookie=0x2b0000000000000c, duration=111.164s, table=2, n_packets=4, n_bytes=
314, idle_age=9, priority=2,in_port=3 actions=output:5,output:6,output:4,outp
ut:1,output:2
 cookie=0x2b00000000000005, duration=117.138s, table=2, n_packets=51, n_byte
=6359, idle_age=2, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000005, duration=117.138s, table=2, n_packets=0, n_bytes=
0, idle_age=117, priority=0 actions=drop
```

Fig. 9 Proactive rules installed by L2 Switch in Switch 20.

```
 cookie=0x2b0000000000000b, duration=112.219s, table=2, n_packets=0, n_bytes=
0, idle_age=112, priority=200,tcp,in_port=6 actions=output:5,output:3,output:
4,output:1,output:2,CONTROLLER:100
 cookie=0x2b0000000000000e, duration=111.943s, table=2, n_packets=0, n_bytes=
0, idle_age=111, priority=200,tcp,in_port=1 actions=output:5,output:6,output:
3,output:4,output:2,CONTROLLER:100
 cookie=0x2b0000000000000d, duration=112.051s, table=2, n_packets=0, n_bytes=
0, idle_age=112, priority=200,tcp,in_port=4 actions=output:5,output:6,output:
3,output:1,output:2,CONTROLLER:100
 cookie=0x2b0000000000000c, duration=112.135s, table=2, n_packets=12, n_bytes
=1078, idle_age=14, priority=200,tcp,in_port=3 actions=output:5,output:6,outp
ut:4,output:1,output:2,CONTROLLER:100
 cookie=0x2b0000000000000a, duration=112.303s, table=2, n_packets=8, n_bytes=
706, idle_age=14, priority=200,tcp,in_port=5 actions=output:6,output:3,output
:4,output:1,output:2,CONTROLLER:100
 cookie=0x2b0000000000000f, duration=111.86s, table=2, n_packets=4, n_bytes=3
28, idle_age=14, priority=200,tcp,in_port=2 actions=output:5,output:6,output:
3,output:4,output:1,CONTROLLER:100
 cookie=0x2c00000000000008, duration=13.238s, table=2, n_packets=11, n_bytes=
830, idle_age=4, priority=5000,tcp,in_port=2,nw_dst=192.168.50.2 actions=outp
ut:3,CONTROLLER:100
 cookie=0x2c00000000000003, duration=13.68s, table=2, n_packets=23, n_bytes=1
889, idle_age=4, priority=5000,tcp,in_port=3,nw_dst=192.168.50.3 actions=outp
ut:5,CONTROLLER:100
 cookie=0x2c00000000000005, duration=13.505s, table=2, n_packets=4, n_bytes=3
10, idle_age=4, priority=5000,tcp,in_port=5,nw_dst=192.168.50.1 actions=outpu
t:3,CONTROLLER:100
 cookie=0x2c00000000000000, duration=13.946s, table=2, n_packets=28, n_bytes=
4902, idle_age=4, priority=5000,tcp,in_port=5,nw_dst=192.168.50.2 actions=out
put:3,CONTROLLER:100
 cookie=0x2c00000000000007, duration=13.321s, table=2, n_packets=10, n_bytes=
1070, idle_age=4, priority=5000,tcp,in_port=2,nw_dst=192.168.50.1 actions=out
put:3,CONTROLLER:100
 cookie=0x2c0000000000000b, duration=12.97s, table=2, n_packets=23, n_bytes=1
926, idle_age=4, priority=5000,tcp,in_port=3,nw_dst=192.168.50.4 actions=outp
ut:2,CONTROLLER:100
```

Fig. 10 MP-TCP Analyzer rules are added in switch 20.

```
controlador1@controlador1:~$ sudo ovs-ofctl dump-flows tcp:192.168.30.2:6633
NXST_FLOW reply (xid=0x4):
 cookie=0x2b00000000000009, duration=188.52s, table=2, n_packets=3, n_bytes=2
54, idle_age=87, priority=2,in_port=5 actions=output:2,output:4,output:3,outp
ut:6,CONTROLLER:65535
 cookie=0x2b00000000000006, duration=188.521s, table=2, n_packets=0, n_bytes=
0, idle_age=188, priority=2,in_port=4 actions=output:2,output:3,output:6,outp
ut:5,CONTROLLER:65535
 cookie=0x2b00000000000008, duration=188.52s, table=2, n_packets=0, n_bytes=0
, idle_age=188, priority=2,in_port=6 actions=output:2,output:4,output:3,outpu
t:5,CONTROLLER:65535
 cookie=0x2b00000000000005, duration=188.521s, table=2, n_packets=3, n_bytes=
254, idle_age=24, priority=2,in_port=2 actions=output:4,output:3,output:6,out
put:5,CONTROLLER:65535
 cookie=0x2b00000000000007, duration=188.52s, table=2, n_packets=6, n_bytes=7
24, idle_age=21, priority=2,in_port=3 actions=output:2,output:4,output:6,outp
ut:5
 cookie=0x2b00000000000003, duration=194.523s, table=2, n_packets=84, n_bytes
=10529, idle_age=4, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000003, duration=194.522s, table=2, n_packets=125, n_byte
s=26418, idle_age=2, priority=0 actions=drop
```

Fig. 11 Proactive rules installed by L2 Switch in Switch 30.

```
 cookie=0x2b00000000000008, duration=189.831s, table=2, n_packets=0, n_bytes=
0, idle_age=189, priority=200,tcp,in_port=6 actions=output:2,output:4,output:
3,output:5,CONTROLLER:100
 cookie=0x2b00000000000009, duration=189.748s, table=2, n_packets=4, n_bytes=
352, idle_age=92, priority=200,tcp,in_port=5 actions=output:2,output:4,output
:3,output:6,CONTROLLER:100
 cookie=0x2b00000000000007, duration=189.923s, table=2, n_packets=12, n_bytes
=1034, idle_age=92, priority=200,tcp,in_port=3 actions=output:2,output:4,outp
ut:6,output:5,CONTROLLER:100
 cookie=0x2b00000000000006, duration=190.007s, table=2, n_packets=0, n_bytes=
0, idle_age=190, priority=200,tcp,in_port=4 actions=output:2,output:3,output:
6,output:5,CONTROLLER:100
 cookie=0x2b00000000000005, duration=190.107s, table=2, n_packets=8, n_bytes=
726, idle_age=92, priority=200,tcp,in_port=2 actions=output:4,output:3,output
:6,output:5,CONTROLLER:100
 cookie=0x2c00000000000001, duration=91.218s, table=2, n_packets=39, n_bytes=
5732, idle_age=82, priority=5000,tcp,in_port=3,nw_dst=192.168.50.2 actions=ou
tput:2,CONTROLLER:100
 cookie=0x2c0000000000000a, duration=90.408s, table=2, n_packets=11, n_bytes=
842, idle_age=82, priority=5000,tcp,in_port=5,nw_dst=192.168.50.4 actions=out
put:3,CONTROLLER:100
 cookie=0x2c00000000000009, duration=90.508s, table=2, n_packets=12, n_bytes=
1084, idle_age=82, priority=5000,tcp,in_port=2,nw_dst=192.168.50.4 actions=ou
tput:3,CONTROLLER:100
 cookie=0x2c00000000000006, duration=90.776s, table=2, n_packets=14, n_bytes=
1380, idle_age=82, priority=5000,tcp,in_port=3,nw_dst=192.168.50.1 actions=ou
tput:5,CONTROLLER:100
 cookie=0x2c00000000000002, duration=91.126s, table=2, n_packets=18, n_bytes=
1479, idle_age=82, priority=5000,tcp,in_port=2,nw_dst=192.168.50.3 actions=ou
tput:3,CONTROLLER:100
 cookie=0x2c00000000000004, duration=90.942s, table=2, n_packets=5, n_bytes=4
10, idle_age=82, priority=5000,tcp,in_port=5,nw_dst=192.168.50.3 actions=outp
ut:3,CONTROLLER:100
```

Fig. 12 MP-TCP Analyzer rules are added in Switch 30.

Examples of reactive rules installed by the Analyzer are presented in red color in Fig. 10 and 12. These rules do matching with the input port, TCP, and destination IP address and do not do flooding, the actions are sending the full packet to an output port and sending packets to the controller.

```
carlos@carlos-HP-Pavilion-dv6-Notebook-PC:~$ sudo ovs-ofctl dump-flows openflow
NXST_FLOW reply (xid=0x4):
 cookie=0x2b00000000000004, duration=257.02s, table=0, n_packets=8, n_bytes=628,
 idle_age=92, priority=2,in_port=3 actions=output:1
 cookie=0x2b00000000000003, duration=257.02s, table=0, n_packets=2, n_bytes=390,
 idle_age=117, priority=2,in_port=1 actions=output:3,CONTROLLER:65535
 cookie=0x2b00000000000002, duration=286.87s, table=0, n_packets=0, n_bytes=0, i
dle_age=286, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
 cookie=0x2b00000000000000, duration=290.838s, table=0, n_packets=126, n_bytes=1
7190, idle_age=3, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000000, duration=290.844s, table=0, n_packets=0, n_bytes=0,
idle_age=290, priority=0 actions=drop
 cookie=0x2b00000000000004, duration=258.688s, table=0, n_packets=24, n_bytes=21
12, idle_age=159, priority=200,tcp,in_port=3 actions=output:1,CONTROLLER:100
 cookie=0x2b00000000000002, duration=286.508s, table=0, n_packets=0, n_bytes=0,
idle_age=286, priority=200,tcp,in_port=2 actions=output:3,output:1,CONTROLLER:10
0
 cookie=0x2b00000000000003, duration=258.872s, table=0, n_packets=0, n_bytes=0,
idle_age=258, priority=200,tcp,in_port=1 actions=output:3,CONTROLLER:100
carlos@carlos-HP-Pavilion-dv6-Notebook-PC:~$
```

Fig. 13. Proactive rules installed by L2 Switch and the Analyzer in Switch 40.

There is not any reactive rule in the OVS (Fig. 13) installed by the Analyzer since the virtual switch never sends any TCP packet to the controller given that the path using OVS is not used, as some ports are marked as *discarding* (both in OVS and Switch 30), hence the Analyzer only gets to install proactive rules.

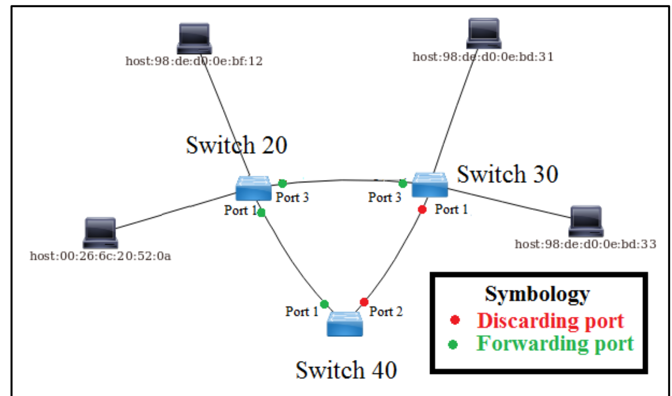

Fig. 14 Network topology in DLUX.

Fig. 14 shows the graphical representation of the topology provided by DLUX for the network being tested. Something to pay attention is that DLUX draws multihomed hosts as if they were different computers, considering the number of interfaces.

Fig. 15 shows an MP-CAPABLE message with its A-H flag bits and the key of the transmitter. Fig. 16 shows the information of a DSS message, including the DSN, the subflow sequence number, flags, data length, ACK and checksum.

```
INFORMATION OF MP-TCP SELECTED MESSAGE
SOURCE MAC ADDRESS:                  98:DE:D0:0E:BF:12
DESTINATION MAC ADDRESS:             00:26:6C:20:52:0A
SOURCE IP ADDRESS:                   192.168.50.1
DESTINATION IP ADDRESS:              192.168.50.3
PROTOCOL:                            TCP
SOURCE PORT:                         36795
DESTINATION PORT:                    21
////////////////////////////////////////////////////////////////////////////////

MP_CAPABLE
MP-TCP SUBTYPE:                      0
MP-TCP VERSION:                      0
LENGTH:                              12
---MP_CAPABLE FLAGS
          1xxx xxxx:     CHECKSUM REQUIRED
          xxxx xxx1:     USE HMAC-SHA1
SENDER'S KEY:                        1546536312253242840
////////////////////////////////////////////////////////////////////////////////
```

Fig. 15 A captured MP-CAPABLE message.

```
INFORMATION OF MP-TCP SELECTED MESSAGE
SOURCE PORT:                         21
DESTINATION PORT:                    36795
////////////////////////////////////////////////////////////////////////////////

DSS
MP-TCP SUBTYPE:                      2
---DATA SEQUENCE SIGNAL FLAGS
          ---x xxx1:     ACK IS INCLUDED
          ---x xx0x:     ACK(4 BYTES) IS INCLUDED
          ---x x1xx:     DSN/SSN/DLL/CHECKSUM ARE INCLUDED
          ---x 0xxx:     DSN(4 BYTES) IS INCLUDED
          ---0 xxxx:     DATA FIN IS NOT INCLUDED
ACK(4 BYTES):                        3072569846
DATA SEQUENCE NUMBER(4 BYTES):       1381440714
SUBFLOW SEQUENCE NUMBER(4 BYTES):    1
DATA-LEVEL LENGHT(2 BYTES):          20
CHECKSUM (2 BYTES):                  59997
////////////////////////////////////////////////////////////////////////////////
```

Fig. 16 A captured DSS message.

B. *Linear topology used for testing*

Fig.17 presents a linear topology using the two HP devices and an OVS. In Fig. 18, the whole network is simulated on a computer running Mininet. The computer uses Ubuntu 14 as the operating system with MP-TCP enabled. Due to the way Mininet is built, the protocols' implementations enabled in Ubuntu will be used during simulation.

Proactive rules do not discard any port for the output actions because this linear topology does not have any loop; besides, when the first TCP packets arrive to the controller, the Analyzer will install reactive rules.

In every case, the expected results were obtained using the Analyzer.

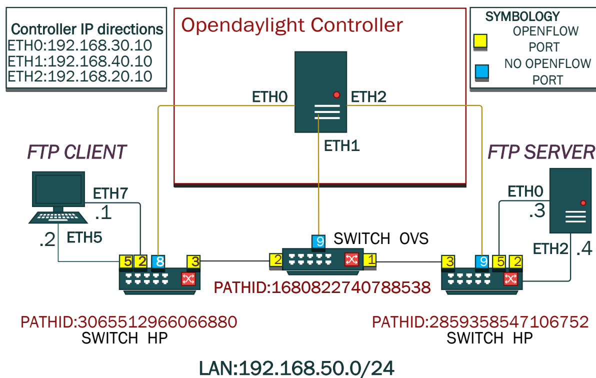

Fig. 17 Linear topology with two physical switches and one OVS.

VI. CONCLUSIONS AND FUTURE WORK

This paper describes several aspects on how a MP-TCP Analyzer was developed using the architecture of SDN and the positive results that were obtained.

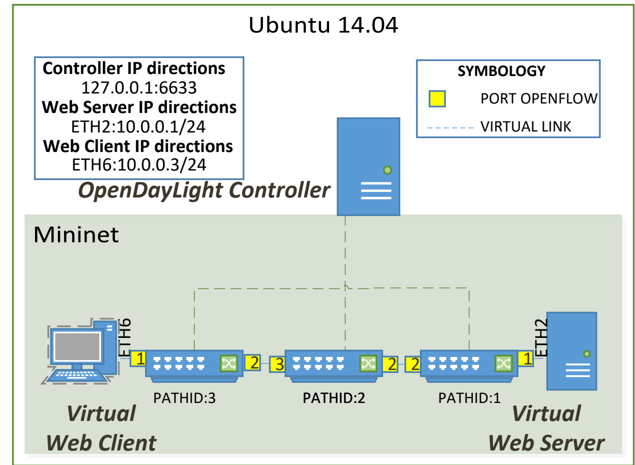

Fig. 18 Linear topology simulated with Mininet using MP-TCP.

The basic idea for deriving TCP traffic to the controller is to install proactive rules that send all TCP packets to the controller and flooding the packets to all other ports. From the received TCP packets in the controller, MP-TCP messages must be discriminated and decoded.

The Analyzer works with topologies that can produce loops by using services provided by OpenDayLight and based on STP. If ports are marked as discarding, they will not be included in the flooding process in proactive rules.

For decreasing overhead and improving performance in the network due to flooding, reactive rules were also used. In any case, what is finally sent to the controller is only 100 bytes that hold MP-TCP messages. However, these bytes allow the analyzer to perform a bit level analysis which provides a degree of detail on MP-TCP messages that is not found in similar solutions using SDN architecture.

One limitation of the Analyzer is CPU consumption due to all the processing done over each packet that is received. If the data rate is higher than 5Mbps in each of the three switches, the controller crashes since it must handle three physical interfaces and more messages per second. The implementation of the Analyzer must be optimized and new tests should be done using machines with more powerful processors and memory.

The Analyzer was proofed to work with a few modules included in OpenDayLight such as L2 Switch without interfering with its functionality. On the other hand, additional tests with different kind of modules should be made since the proactive and reactive rules installed by the Analyzer may become incompatible with them and alternatives to overcome these problems should be proposed.

Finally, the Analyzer could be the basis to develop other modules for OpenDayLight related to MP-TCP such as the one presented in [8] and code for additional protocols could be added.

ACKNOWLEDGMENT

The authors would like to thank our university, Escuela Politécnica Nacional, for supporting our research.

REFERENCES

[1] C. Raiciu, C. Paasch, et al, "How hard can it be? Designing and implementing a deployable multipath TCP," *Usenix.,* vol.0, pp. 29–29, April 2012.

[2] MD-SAL: L2 Switch, opendaylight wiki, https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:L2_Switch/

[3] Allied Telesis, *Mirroring, Feature Overview and Configuration Guide*, United States of America, 2016, pp.3-5.

[4] D. Jhatakia, *Network Packet Monitoring Optimizations in Data Centre*, United States of America: Happiest Mind Technologies, pp.4-9.

[5] Taps, Bypass and NTO Solutions, Empowered, https://empowerednetworks.com/visibility/network-visibility.

[6] Using SDN to Create a Packet Monitoring System, NetworkWorld, http://www.networkworld.com/article/2226003/cisco-subnet/using-sdn-to-create-a-packet-monitoring-system.html.

[7] Skydive: a real-time network analyzer, RDO, https://blogs.rdoproject.org/7874/skydive-a-real-time-network-analyzer.

[8] M. Sandri, A. Silva, L. Rocha, F. Verdi, "On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks," *IEEE Trans. Magn.* , vol. 0 , pp. 9-16, March 2015.

[9] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF RFC, 2013.

[10] I. Bernal "A Friendly Introduction to the Requirements and Supporting Technologies for 5G Cellular Networks," *Revista Politécnica*, vol. 37, No. 1, March 2016.

[11] What is Openflow, A. Bregma, http://abregman.com/2016/11/30/openflow-introduction/

[12] J. Medved, R. Varga, A. Tkacik, K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," *IEEE Trans. Magn.*, vol. 0, pp. 1-6. June 2014.

[13] Linux Foundation, *OpenDaylight Developer Guide*, 1st ed., San Francisco: Linux Foundation, 2015, pp.40-50.

[14] Installing OpenDaylight, OpenDaylight Project, http://docs.opendaylight.org/en/stable-boron/getting-started.

[15] SDNHub Opendaylight Tutorial, SDNHub, https://github.com/sdnhub/SDNHub_Opendaylight_Tutorial guide/installing_opendaylight.html#install-the-karaf-features

[16] Open vSwitch, Linux Foundation, http://openvswitch.org/