A study on **Software Defined Networks**

Traffic Engineering

Master Thesis

Postgraduate programme “Technology Education & Digital Systems” – Digital Communications and networks

Charalampos Patras, University Of Piraeus 2013-2015

Supervisor: Prof. Panagiotis Demestichas

Dedicated to my parents

and my sister

ACKNOWLEDGEMENTS

I would like to thank my advisor, Konstantinos Tsagkaris for guiding and supporting me. You have set an example of excellence as a researcher, mentor, instructor, and role model. Also I would like to thank Mr. Prof. Panagiotis Demestichas for the support that he dedicated to me.

I would especially like to thank my amazing family for the love, support, and constant encouragement I have gotten over the years. In particular, I would like to thank my parents and my sister Helen. You are the salt of the earth, and I undoubtedly could not have done this without you.

Table of Contents

List of Figures

List of Tables

Acronym analysis

SDN:	Software Defined Networks
LSPs:	Label-switched Paths
MST:	Minimum Spanning Tree

Περίληψη

Ο σκοπός αυτής της διατριβής είναι η ανάπτυξη και παρουσίαση ενός προγράμματος λογισμικού (εφαρμογής), που εφαρμόζεται σε έναν controller δικτύου ηλεκτρονικών υπολογιστών, με στόχο την δρομολόγηση των δεδομένων στο δίκτυο με ελάχιστο κόστος και με τη λειτουργία συνεχούς ενεργοποίησης (on / off) των προϊόντων του το δίκτυο. Βελτιστοποίηση του δικτύου, επιλέγοντας το σωστό μονοπάτι δεδομένων (LSP) βασίζεται στην εφαρμογή του αλγορίθμου Kruskal (greedy algorithm). Για την ανάπτυξη αυτής της εφαρμογής χρησιμοποιείται ένα ερευνητικό πρωτόκολλο επικοινωνίας δικτύων, το OpenFlow. Η χρήση του OpenFlow είναι σημαντική διότι έχει τη δυνατότητα χωρισμού του πεδίου δεδομένων (data) και τον έλεγχο του δικτύου (network control). Για την υλοποίηση αυτή χρησιμοποιείτε ο controller Floodlight, έτσι υπάρχει ένα δίκτυο, που βασίζει τη λειτουργία του σε αυτόν τον controller, και όχι σε μεμονωμένες συσκευές (switches, routers κτλ.). Επιπλέον, χρησιμοποιείται το λογισμικό Mininet για να δημιουργηθεί η τοπολογία του δικτύου. Η εφαρμογή υλοποιείται σε γλώσσα προγραμματισμού Java. Το λογισμικό που χρησιμοποιείται αποτελεί ένα εξομοιωτή δικτύου. Με τη βοήθεια αυτού του λογισμικού δημιουργείται ένα σύνολο τερματικών (hosts), δρομολογητών (routers), διακοπτών (switches) και άλλα αντίστοιχα links Ethernet σε ένα ενιαίο πυρήνα του Linux (kernel).

Λέξεις κλειδιά: SDN, openflow, mininet, traffic engineering, kruskal, minimum spanning tree

Abstract

The aim of this thesis is the development and presentation of a software program (application) that is applied to a computer network controller, aimed at routing the data to the network with minimal costs and the continuous switching operation (on / off) of devices of the network. Optimization of the network by choosing the right data path is based on the application of Kruskal algorithm. For the development of this application a research communications protocol is used, ie OpenFlow. Using OpenFlow is important because it offers the possibility of disconnection of the data field and the network control. The Floodlight controller protocol is used and there is a centralized network that bases its operation on a central controller, rather than to individual devices. In addition Mininet software is used to create the network topology. The application is implemented in Java programming language. The software used constitutes a network emulator. It can simultaneously perform a set of terminals, routers, switches and other Ethernet respective links on a single Linux Kernel (core).

Key words: SDN, openflow, mininet, traffic engineering, kruskal, minimum spanning tree

1. Introduction

Currently, the use and exploitation of the Internet is growing rapidly. Network computing and IT devices have become an integral part of every day life. This has resulted in phenomenal growth of network infrastructure and waste of energy consumed by the network devices. The need to find energy and cost saving solutions is a reflection that concerns the computing community. The solutions and methods can be found at different levels of the network structure, but most important of them is the same of the infrastructure, which consists of all the devices, such as routers, switches, hubs, servers and clients.

Routing of network devices to this day remains static. Routers and switches create routes to the network using various algorithms (eg minimum distance - Dijkstra) for communications between the various devices. These algorithms are constructed so as to choose the shortest path. Nowadays, when there is a variation in connection speeds and applications with different speed needs, these methods are considered unorthodox as they may lead to a large increase of load processed in one device while another remains inactive.

This thesis aims to investigate a mechanism to dynamically control the routing of packets calculating periodically the optimized route. We use a new architecture, OpenFlow, which provides the ability to create virtual networking infrastructures to routers and create multiple paths to a physical network without being necessary to re-run connections creation algorithm for each path from the router.

The second Chapter presents OpenFlow technology. The structure of the architecture and the reasons for which it was developed are presented. Then we presents variety of controllers that have been developed. This Chapter also includes a description of Mininet software for network topology creation.

The third chapter deals with the path control process through which the traffic is handled in the network, or else Traffic Engineering. Managers wish to influence the

characteristics of a path for optimization of network resources. The aim is to avoid the situation of certain parts congestion when others are underutilized.
The fourth Chapter describes Kruskal algorithm, the basis of traffic engineering in our application. Kruskal algorithm is analyzed and compared with other optimization algorithms. Several theoretical examples are presented for better understanding of Kruskal operation.

The fifth Chapter provides a description of the implemented application, developed in Java programming language. The software used for creating the virtual topology and performing the minimum spanning tree (MST) function is described. Additionally there is a presentation and explanation of topology used in this work. The sixth Chapter provides the tests and results of MST package.

2. Software Defined Networks

The Internet as we know it today, shows limitations due to the widespread and rapid expansion, thus limiting the scope for developing and implementing innovations. The "software-defined" networks (Software-Defined Networks) are the future of computer networking. This architecture provides researchers an easier method to test new technologies and protocols. The most important thing is that SDN can be a main substrate for Cloud Computer networking (Cloud Computing) [1].

The SDN is a new emerging computer network architectures. In a computer network we have the concepts of data plane and control plane. Today, the interface between the control plane and the data plane is closed and is inside routers and switches so that no one can easily change the routing protocols used on a computer network. The basic idea of the SDN architecture is the decoupling of the control plane from the data plane and the creation of an open interface between them. The control plane runs outside of the routers over a so-called network operating system (NOS), which manages the forwarding tables of the routers and switches of a network. This approach is much easier to implement innovative routing and traffic management techniques since a new routing protocol can be implemented very quickly, simply by using new software over the NOS, without requiring changes to routers and switches. The SDN architecture has approached much interest from industry in the last 2-3 years, already supported by many companies producing routers and switches, such as Cisco and Juniper, and is already used in some networks, such as Google inter-data-center network [1].

An important role in this direction plays the OpenFlow protocol. Via OpenFlow, the separation can be achieved between the level of control and forwarding packets in a network. Additionally, using the FlowVisor the discriminant policy of network resources can be achieved to isolated fragments [2].

Today, each of the manufacturers of computer networking devices allows a different degree of planning and control of routers and switches from administrators. This often leads to reduced usability of these devices, as well as heterogeneity in the management mechanisms of network traffic from different manufacturers devices.

Furthermore, the management of computer networks requires customization, separately for each device used [2].

The basic idea is that we can exploit the fact that most Ethernet switches (switches) now contain flow record data tables (flow-tables) required to implement services of Network Address Translation (NAT), Quality of Service (QoS), Firewall, etc.. These tables are implemented using multiple Ternary Content Addressable Memories (TCAMs), containing access-lists to filter packets based on the MAC address, and QoS access-lists for the priority of network traffic. OpenFlow provides a free protocol for the programming of these flow-tables. Each OpenFlow switch is controlled by a researcher or by the network administrator through a Controller (Controller). A key feature of the Controller is the fact that it can add or remove data flows (flows) in the flow-table of OpenFlow switch. Finally, using the FlowVisor we can create isolated network resources (slices), each of which will be controlled by a particular Controller [2].

With these methods researchers perform experiments in heterogeneous switches or routers (routers). But it is important that this is achieved without requiring manufacturers to expose the inner workings of their products. In the following paragraphs we will explain in detail these methods, based on version OpenFlow Protocol [1].

Examples of SDN uses

There are a lot of cases that Software Defined Networks can find application. We are going to report some of them and we will describe the cases. Some of them are Network Access Control, Network Virtualization, Virtual Customer Edge, Dynamic Interconnects, Virtual Core and Aggregation, Datacenter Optimization

- Network Access Control

It is an ability that sets the appropriate privileges for any user or device of the network. It controls how someone accessing the networks, including access control limits, and the incorporation of service chains as well as appropriate quality of service (QoS).

- Network Virtualization (NV)

It is an ability to create a virtual network on top of a physical network, allowing a large number of multi-tenant networks to run over a physical network, spanning multiple racks in the datacenter or locations if necessary, including fine-grained controls and isolation as well as insertion of acceleration or security services.

- Virtual Customer Edge

It is the ability to virtualize the customer edge either through creation of a virtualized platform on customer premises or by pulling in the functions closer to the core on a virtualized multi-tenant platform hosted either in a carrier point-of-presence, regional datacenter, central datacenter.

- Dynamic Interconnects

It is the ability that creates dynamic links between locations, including between DCs, enterprise and DCs, as well as dynamically applying appropriate QoS for those links.

- Virtual Core and Aggregation

It is the ability that virtualized core systems for service providers including support infrastructure, as well as dynamic mobile backhaul.

- Data Center Optimization

It is the ability that uses SDN and NFV, optimizing networks to improve application performance by detecting and taking into account affinities, this it is made by orchestrating workloads with networking configuration.

2.1 Openflow Protocol

An OpenFlow switch consists of a flow-table, which is used for mapping and packet forwarding, and a secure communication channel (secure channel) to a Controller. Finally, the Controller manages the OpenFlow switch through the secure channel using the OpenFlow protocol (Figure 1).

The flow-table contains entries for flows, counters and actions for each record. Each packet that enters the OpenFlow switch is checked against the records of flow-table. In the case it matches with any record, the actions that accompany this registration are applied, otherwise the packet is forwarded on the Controller via the secure channel. The Controller is now responsible to reach a decision for the route that will follow the pack, adding or removing entries from the flow-table of the switch [3].

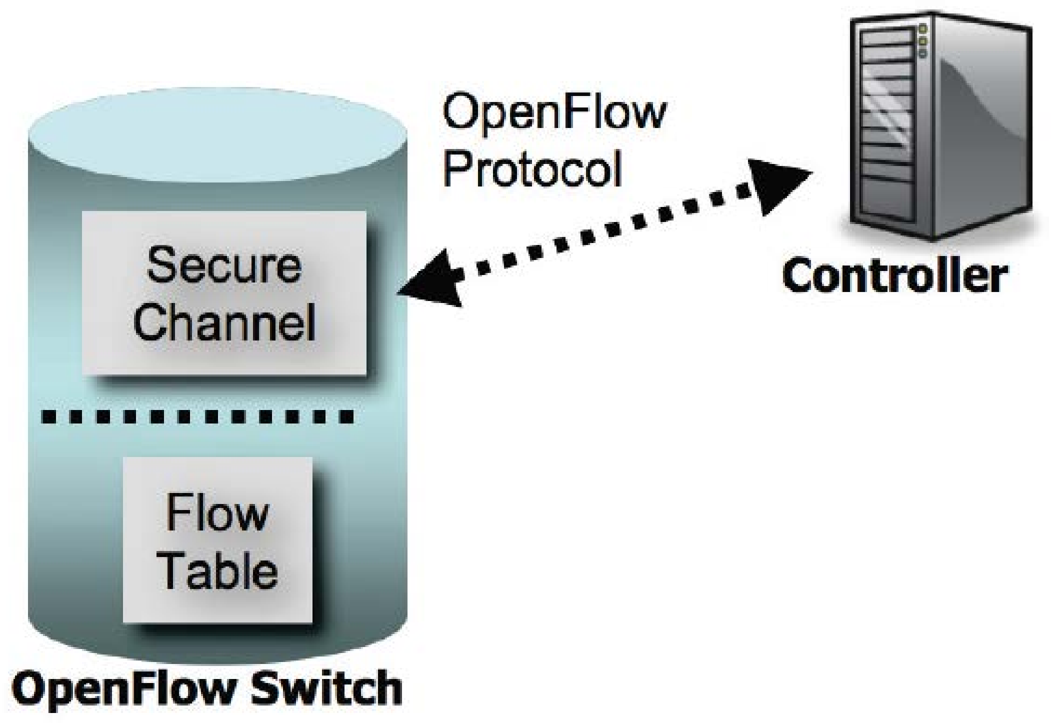
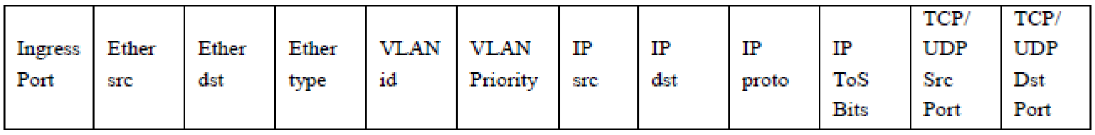


Figure 1. OpenFlow switch that communicates with Controller via a secure channel using OpenFlow protocol

Generally, the OpenFlow protocol was created to provide a standard way of OpenFlow switch communication with the Controller. This protocol supports three types of messages, Controller-to-switch, asynchronous (asynchronous), and symmetric (symmetric). Still, for every kind of message we can distinguish several subcategories, which are detailed below. The Controller-to-switch messages are the source of the Controller and allow to directly manage, or supervise, the state of a switch. The asynchronous messages start from the switch and its purpose is to inform the Controller or events (events) that occur in the network, or changes in the status of soitch. Finally, symmetrical messages from either the Controller or the switch and sent without before to have been a request [3].

2.1.1 Flow-table

Each entry of the flow-table contains header fields which are contrasted with the corresponding fields of each packet entering the OpenFlow switch. It also contains counters that are updated each time a packet is assigned to a particular record. Finally containing actions to be implemented in case of packet matching with some record.



Ingress Port	Ether src	Ether dst	Ether type	VLAN id	VLAN Priority	IP src	IP dst	IP proto	IP ToS Bits	TCP/ UDP Src Port	TCP/ UDP Dst Port

Table 1. Information constitute a criterion for assigning a packet to a particular flow are included in the records of the flow-table

In Table 1 and twelve header fields are shown that can be stored in each record of the flow-table. These twelve fields will be compared with the corresponding fields of each packet entering the OpenFlow switch. It is worth noting that every field can have either a specific value or the value "ANY" so and compare with the corresponding field of a packet will always be true [3].

In each OpenFlow switch, counters are maintained for each table for flows, and for switch ports, as well as for queues. Table 2 shows the total counters held by an OpenFlow switch. Each entry flow is associated with a list of actions (this list can contain from zero to any number of actions). The actions contained in such a record indicating how the switch handles packages that will respond to that entry. If there is no forward action in the list, then the packet is discarded (drop).

A switch can reject the creation of a new registration flow in case it can not edit the list of actions it encompasses. Considering this situation, an OpenFlow switch is not required to support all kinds of actions supporting the OpenFlow protocol (Table 3), but must support at least those marked "REQUIRED", which are both necessary for the realization of the basic functions. When a switch is connected to the Controller, then it gives information about the optional actions (display "OPTIONAL" in table 3) supported [4].

The basic set of OpenFlow switch actions, based on which packet forwarding is implemented, is the FORWARD set. Each switch must promote a package to any physical port, and with these virtual ports:

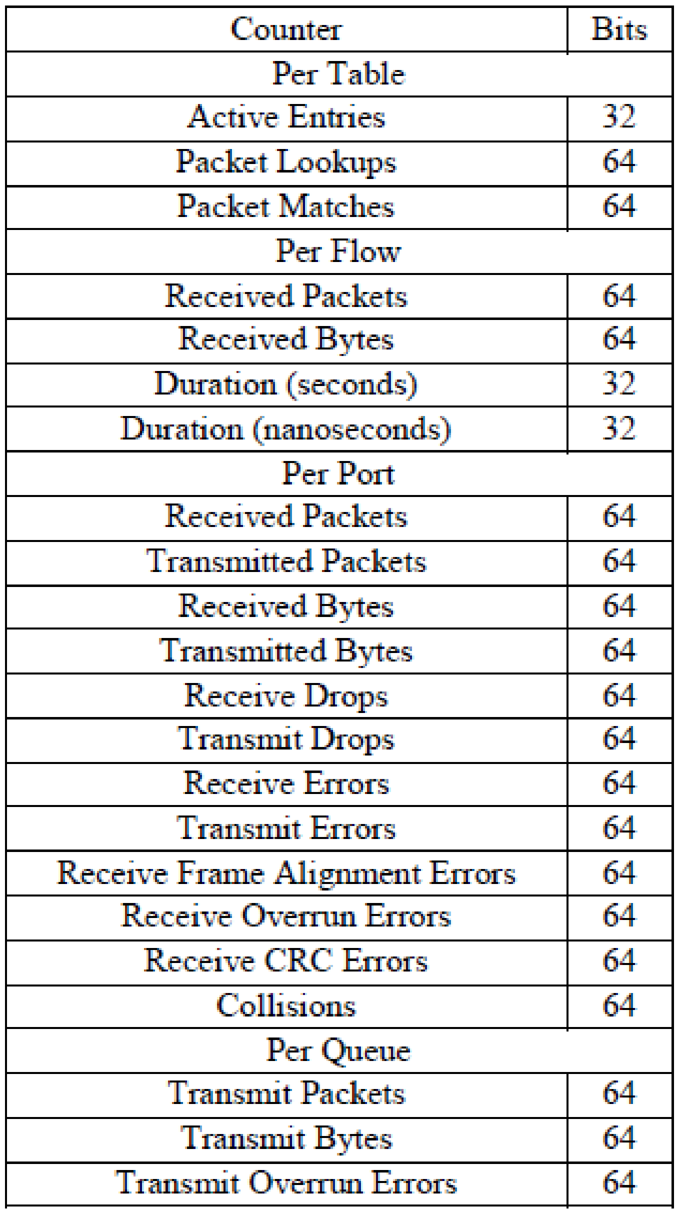
- ALL: Promoting a packet to all interfaces of the switch, except the input interface.
- CONTROLLER: Send 128 bits (or whole) of the package to the Controller
- LOCAL: Send the package to the networking stack's own switch
- TABLE: It performs some actions in flow-table of the same switch. Action concerns the only packet-out messages
- IN_PORT: Promotes the package from the entrance door

Besides these five virtual doors, a switch can optionally supports the following two:

- NORMAL: packet forwarding with the "normal" way forward packets (traditional forwarding path) that supports each switch
- FLOOD: promotion package with the minimum Spanning Tree, without including the port of entry of the package

Generally, the OpenFlow switch is divided into the following two categories:

- OpenFlow-only or Dedicated OpenFlow switch: A "spineless" datapath which forwards packets according to the suggestions the Controller controlling it.
- OpenFlow-enabled switch: Specifically commercially switch, routers and access points modified with the addition of OpenFlow firmware. Firmware consists of the Flow Table, the Secure Channel, and the OpenFlow Protocol.



Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

Table 2. List of available counters

A critical difference between the two is that the Dedicated OpenFlow switch only supports FORWARD actions of Table 3 labeled REQUIRED. Instead, OpenFlow-enabled can support the FORWARD action NORMAL. Both types, however, can support the action FLOOD.

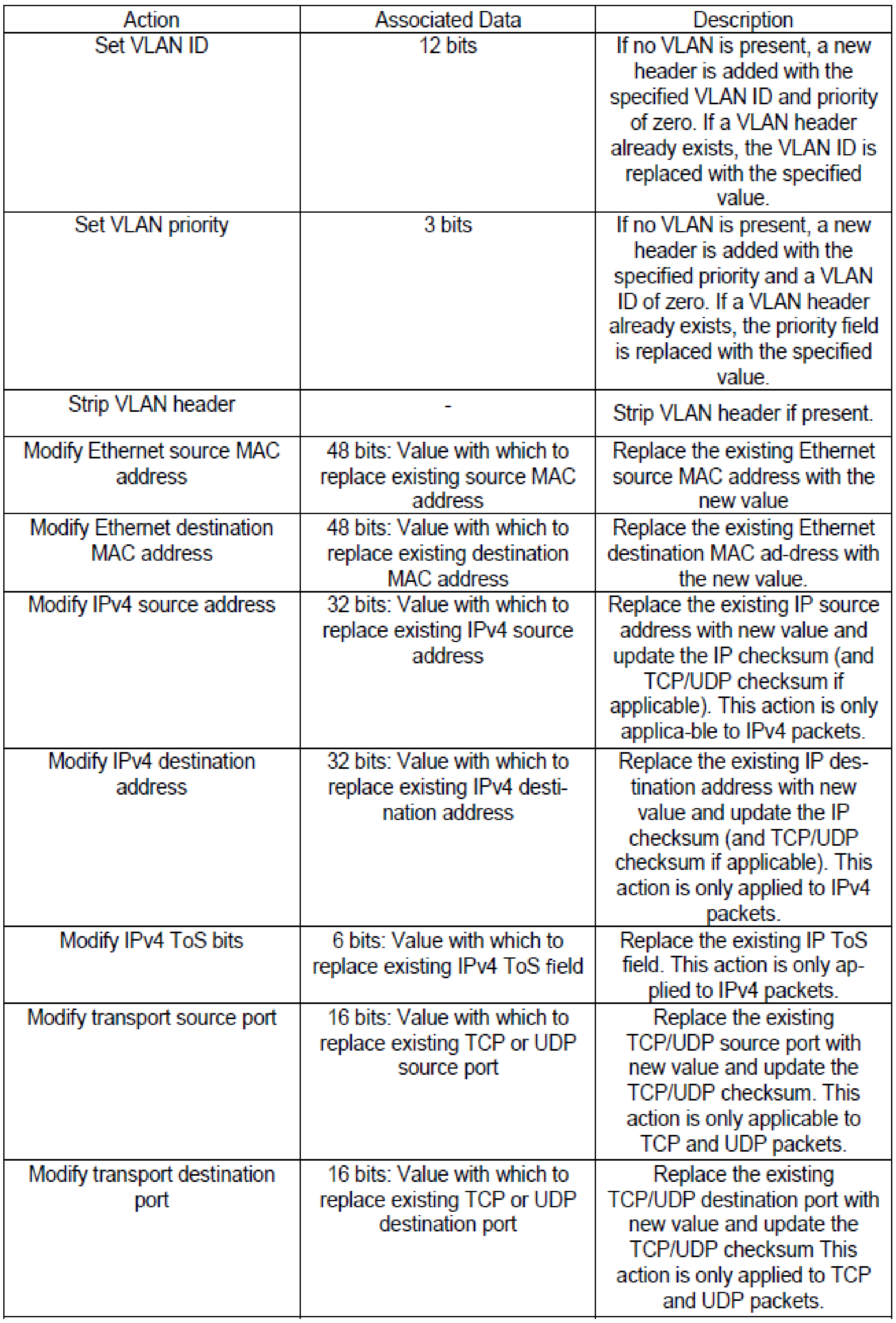
Perhaps the most important set of "voluntary" actions is the "Modify-Field" as it may change the value of a packet header, greatly increasing the usefulness, and improving the functioning of the OpenFlow switch. Table 4 shows the total "Modify-Field" actions.

Action	Function	Necessity
FORWARD	Forward packet to any physical port, and to the visuals below: • ALL • CONTROLLER • LOCAL • TABLE • IN_PORT	REQUIRED
FORWARD	Packet forwarding to ports: • NORMAL • FLOOD	OPTIONAL
ENQUEUE	Packet forwarding to port tail	OPTIONAL
DROP	A flow profile without any action specified, indicates that the packet should be rejected	REQUIRED
MODIFY-FIELD	Enables an OpenFlow soitch alter the values of the header of a packet	OPTIONAL

Table 3. Set of actions that can be supported by an OpenFlow switch

2.1.2 Secure Channel

The Secure Channel is one interface (interface) connecting each OpenFlow switch to a Controller. Through this interface, the Controller regulates and manages the switch, informed of events (events) via the switch and sends packets through it. The interface may vary depending on the implementation of OpenFlow switch, but each message destined to be sent through the secure-channel must be standardized with OpenFlow protocol [3].



Action	Associated Data	Description
Set VLAN ID	12 bits	If no VLAN is present, a new header is added with the specified VLAN ID and priority of zero. If a VLAN header already exists, the VLAN ID is replaced with the specified value.
Set VLAN priority	3 bits	If no VLAN is present, a new header is added with the specified priority and a VLAN ID of zero. If a VLAN header already exists, the priority field is replaced with the specified value.
Strip VLAN header	-	Strip VLAN header if present.
Modify Ethernet source MAC address	48 bits: Value with which to replace existing source MAC address	Replace the existing Ethernet source MAC address with the new value
Modify Ethernet destination MAC address	48 bits: Value with which to replace existing destination MAC address	Replace the existing Ethernet destination MAC ad-dress with the new value.
Modify IPv4 source address	32 bits: Value with which to replace existing IPv4 source address	Replace the existing IP source address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applica-ble to IPv4 packets.
Modify IPv4 destination address	32 bits: Value with which to replace existing IPv4 destination address	Replace the existing IP destination address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applied to IPv4 packets.
Modify IPv4 ToS bits	6 bits: Value with which to replace existing IPv4 ToS field	Replace the existing IP ToS field. This action is only applied to IPv4 packets.
Modify transport source port	16 bits: Value with which to replace existing TCP or UDP source port	Replace the existing TCP/UDP source port with new value and update the TCP/UDP checksum. This action is only applicable to TCP and UDP packets.
Modify transport destination port	16 bits: Value with which to replace existing TCP or UDP destination port	Replace the existing TCP/UDP destination port with new value and update the TCP/UDP checksum This action is only applied to TCP and UDP packets.

Table 4. Field-Modify actions

2.1.3 Controller-to-switch messages

Messages of Controller-to-switch type start from the Controller to the switch, without being required to send a message in response. These messages are divided into the following subcategories [3]:

- Features: Since the start of the session between the Controller and the switch through Transport Layer Security (TLS), the Controller sends a message to the switch requesting information on capabilities (features request), and expects a relative answer (features reply).
- Configuration: The Controller can configure the switch settings that controls, or request information about them. In this case the switch is required to respond with a note.
- Modify-State: These messages are used by the Controller mainly for addition, deletion, or modification of the flow-table that exists in the switch, or to adjust the properties of the ports.
- Read-State: Used by the Controller to gather statistics on the table of flows, the ports, as well as for each record flow separately.
- Send-Packet: The send-packet messages are used to indicate the Controller to switch via which specific port to promote a package.
- Barrier: Messages Barrier request / reply are used to confirm the Controller that the requirements for a particular message apply. They are even used to ascertain the Controller for the completion of a process.

2.1.3 Asynchronous messages

The asynchronous messages are sent by the switch, without first having been requested by the Controller. Their purpose is to inform the controller of packet arrivals, changes in the state of switch, or an error that has occurred [3].

The four main sub asynchronous messaging types are:

- Packet-in: Any new package that enters the switch and is not mapped to any existing entries flow, causes the creation and sending a message Packet-in to the Controller (packet-in event). If the switch has enough available memory to cache (buffer) this package, then the message that will be sent will contain 128 bytes with the necessary information that the Controller. The information relates to the values of the header of the packet entered, and a one identification value (buffer ID) of the package. If the switch does not support caching packets, or does not have enough available memory, then the message to be sent on the Controller will include the entire original package.

- Flow-removed: When a record is added to the flow switch from the Controller via a flow-modify message, dictated the switch after how much idle time should erase that entry. It is even dictated when to shut off the general, regardless of the activity associated with this entry. Simultaneously dictate the switch if you should notify the Controller after such a deletion, which is a message-type flow-removed.
- Port-status: To switch uses these messages in cases of change of state of a port, for example, if a user disables a specific switch port. Additionally, it is used in cases of change in state of a port as defined by Protocol 802.1D.
- Error: With these messages, the switch can inform the Controller for problems or errors that may arise.

2.1.4 Symmetric messages

Symmetrical messages may be sent either by a switch, or a Controller, without the other party having requested such an action, and are separated into the following three categories [3]:

- Hello: Messages of this type are exchanged between the switch and the Controller the moment you finish the connection between them.
- Echo: Posts of type echo request / reply can be sent either sides and is used for measurements of delay (latency) or frequency range (bandwidth). It also is used to verify whether the connection between them is active.
- Vendor: The purpose of these messages is to provide a space for further functionality, those for types of OpenFlow messages. They have been implemented primarily for future versions of OpenFlow.

2.1.5 Switch – Controller connection & Encryption

The switch must be able to establish communication with the Controller, via an IP address and a port, which are set by the user of the switch and remain stable. The movement through the secure channel is not checked against entries in the table of flows, and for this reason, the switch must recognize the rest of network traffic as local, in order to compare with recordings of the flow-table [4].

The communication between the switch and the Controller is via a connection TLS. Linking this initiates the switch to the Controller, who usually expects this to TCP port 6633. Then the authentication takes place via exchange certificates with private key. For this reason, each switch must have a certificate certifying the Controller and another for his certification to the Controller [4].

If after some time the connection is lost, the switch attempts to connect with him, or spare Controllers that have been defined. If this connection fails after a specified number of attempts, the switch automatically enters a safety mode (emergency state) and returns a list of flows to its original state (reset). From that moment, and until it again reaches its connection to a Controller, the mapping and forward process of packets entering the switch, dictated by a flow table security (emergency), which will be determined by the user. So, once recovered the connection to a Controller is possible to maintain or abolish these safety records [4].

2.1.6 Read State messages

Messages of type Read-State, used by the auditor to gather statistics from the router. Finding the desired path in the routing algorithm, it is necessary to collect statistical data on various network elements and store these data into a database that will be updated frequently [4].

The OpenFlow protocol enables such data collection through Read-State. More specifically, the controller sends at regular intervals messages of type OFPT_STATS_REQUEST to the network devices. Routers respond with one or more messages of type OFPT_STATS_REPLY.

The only value specified as flag in a reply message is the value 0x0001, and is defined only if you follow more than one replies. To facilitate implementation, the routers may send replies without additional entries. However, they should always be sent after a message containing flag field values. The identities of the transactions (xid) replies must always match the request. Both in requests and replies, the field type determines the type of information provided and determines how the field is interpreted.

In all types of statistics, if a counter value is not available on the router, its value is set to -1.

2.1.6 Assigning Packages - Flow Registration

For each new packet received, the OpenFlow switch performs the procedure shown in Figure 2, to decide how to manage it and where to promote it. Of course, the control / comparison process of the header of the new packet (Figure 3) depends on the type of package, as described below [5].

- Rules that are characterized by a specific port of entry, contrasted with the natural port that received the packet.
- The Ethernet headers, as shown in Table 2, are used for all packages.
- If the packet has VLAN tag (that is type 0x8100), then the VLAN ID and PCP fields take part in the match.
- For ARP packages (Ethernet type 0x8606), the matching process can be used and source IP addresses and destination.
- For IP packets (Ethernet type 0x8000), the matching fields include those of the IP header.
- For IP packets using TCP or UDP (IP protocol 6 or 17), in matching the transport ports are used.
- For IP packets using ICMP protocol (IP protocol 1) in pairing Type and Code fields are used.
- For IP packets with non-zero fragment offset, or more Fragments bit set, the transport ports are considered to be zero for the mapping process.

If the values of the packet headers match those specified in any recorded flow, then the packet is assigned to the particular recording. Then, if the packet after the comparison with the records of the flow-table is matched with some records, then the meters comprising the particular record are renewed. If the package is not assigned any entry, then it is forwarded to the Controller via the secure channel.

Finally, it is important to remember that packets are mapped to flow entries based on some priority. Each package can be paired with multiple entries, but the order of matching is dictated by the priority of registration. Thus, a record that exactly matches the headers of a packet (that is, for example it contains no wildcards), always has the highest priority. If for a package, multiple table records of flows match and have the same priority, then the switch is free to choose any string for package matching [5].

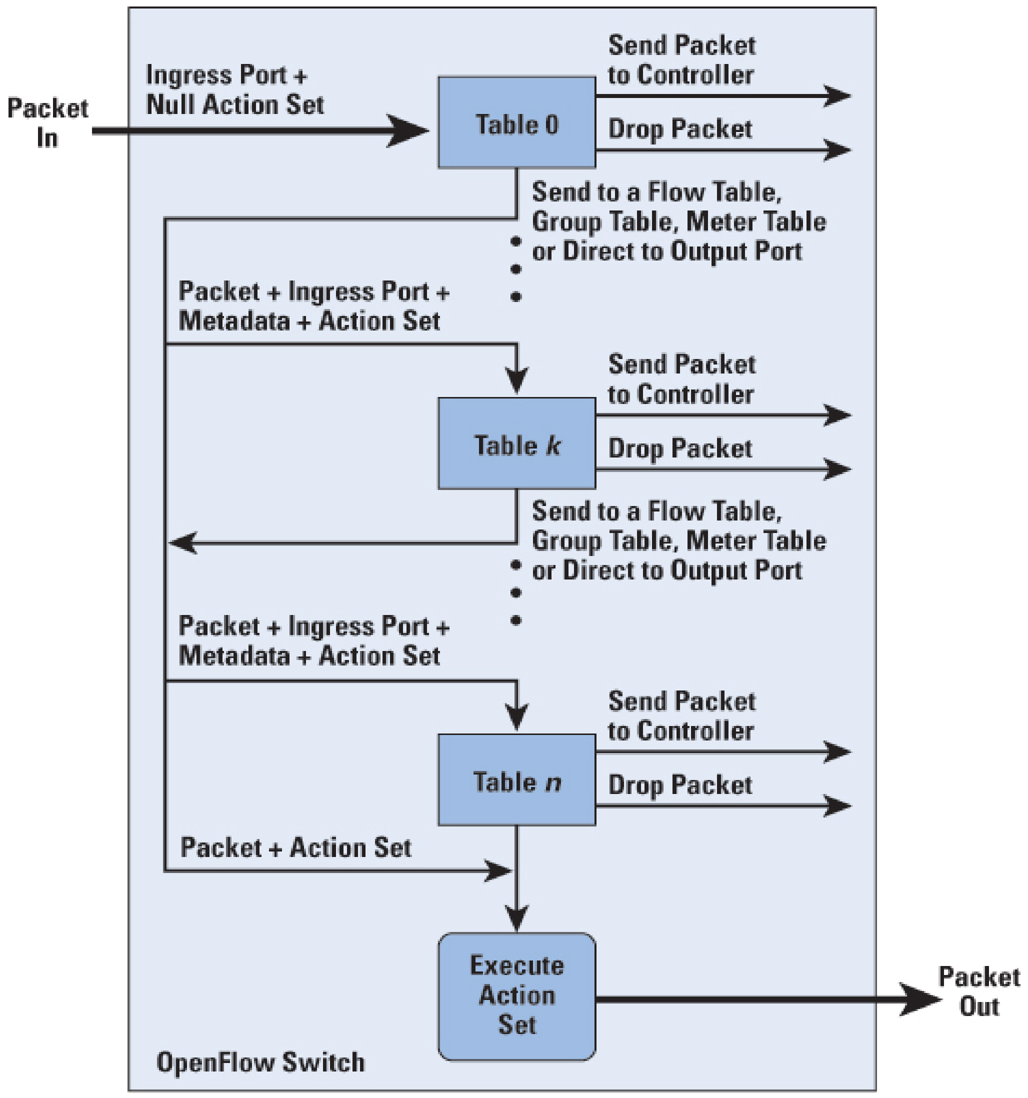

Packet In
Ingress Port + Null Action Set
Table 0
Send Packet to Controller
Drop Packet
Send to a Flow Table, Group Table, Meter Table or Direct to Output Port
Packet + Ingress Port + Metadata + Action Set
Table k
Send Packet to Controller
Drop Packet
Send to a Flow Table, Group Table, Meter Table or Direct to Output Port
Packet + Ingress Port + Metadata + Action Set
Table n
Send Packet to Controller
Drop Packet
Packet + Action Set
Execute Action Set
Packet Out
OpenFlow Switch

Figure 2. Packet Flow in an OpenFlow switch

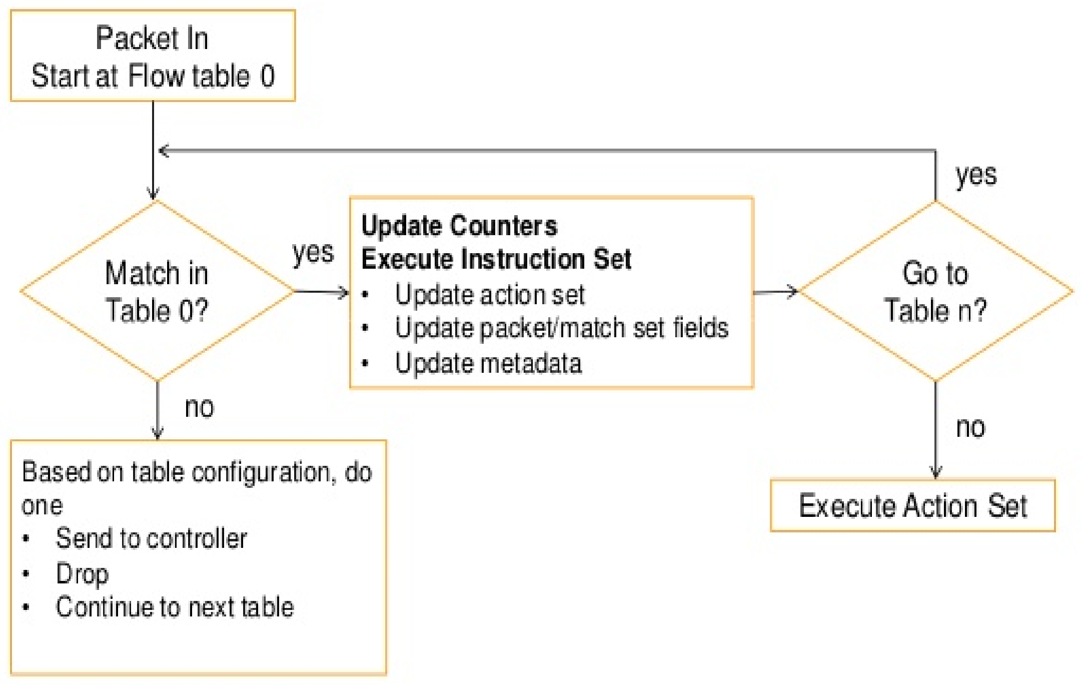


Figure 3. Packet matching with Flow Table in OpenFlow

2.2 Controller

Until recently, the management of computer networks was exclusively implemented through low-level settings separately for each network element (eg, switch, router, etc.). This practice is highly dependent on the physical topology of each network. Plus, by separating the Data Plane from the Control Plane in OpenFlow protocol we can talk about a Computer Network Operating System that provides a single, centralized programming environment to control an entire network.

Such an operating system does not manage the network itself, but provides the ability to monitor and control the processes, through the integrated programming environment. So the management of the network can now be done by applications built and "run" in the operating system. So two innovations in networks management are changing the way it was done so far. Firstly the system supports applications through a central programming model, which can manage an entire network of computers. Second, these applications can be written to a higher level of generalization (eg user and host names) rather than in low-level configurations (eg IP addresses or MAC). All you need is the operating system that maintains mappings between these two levels [5].

Thus the concept of the Controller was created, which is nothing more than a Computer Operating System Network, responsible for their management. As already mentioned, each OpenFlow switch is operated by a Controller. The Controller is responsible for taking decisions on the state of the switch and the promotion of new packages that enter into it. Thus a Controller that can be run on a single PC is able to handle multiple OpenFlow switches and to have an optical entire network that controls (Figure 4).

There is currently a limited but growing number of Controllers that we can use, but here we will focus on NOX and Beacon Controllers, which are the most widespread.

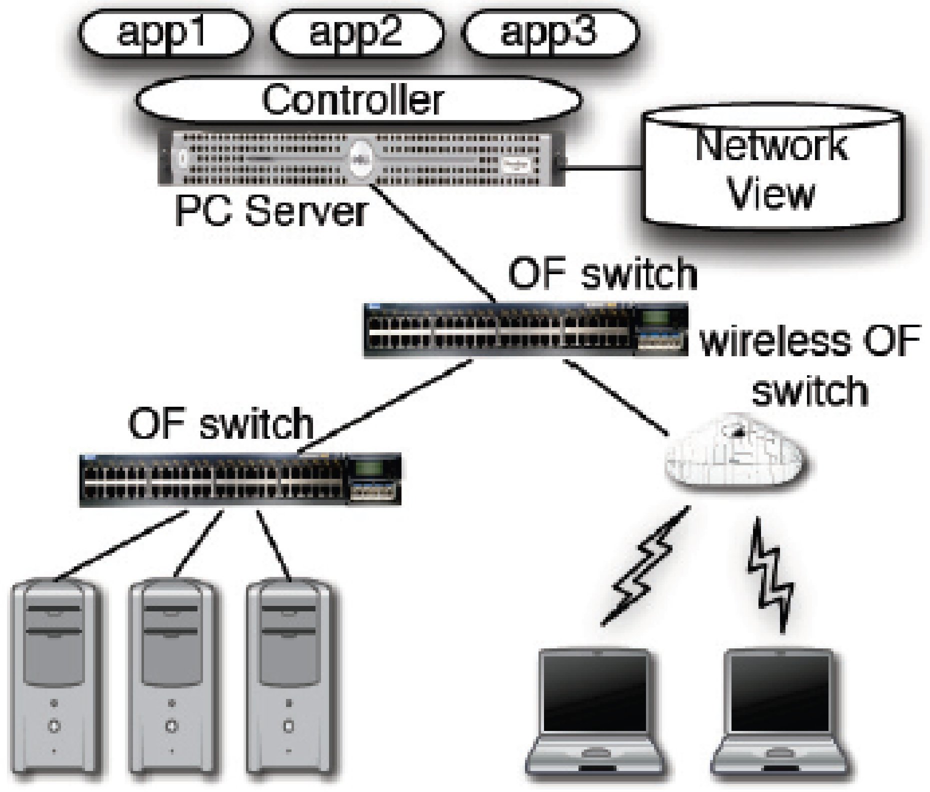


Figure 4. A Controller manages multiple OpenFlow switches

2.2.1 NOX Controller

NOX Controller is a control platform and a computer network and provides a high-level programming environment. This way it can create applications which could be used by NOX to make decisions for the management and monitoring of the network. The main purpose of these applications is to decide how and whether to launch each package on a computer network, which is achieved by using flows [3].

Analyzing the network level flows, if some decision has been made for the packet, the packets that will follow and have the same headers, could be treated in the same way without intervention of the Controller. Thus, the NOX can be managed by a small network of several hosts, or a large network of hundreds of switches, providing in both cases an easily modifiable way to control these networks [4].

All that NOX requires to function is to have been connected with at least one OpenFlow switch, using the OpenFlow protocol. As already mentioned in the previous paragraph, if a new packet enters the switch and is not associated with any of the existing entries in the table of flows, it will be sent to NOX. From this point onwards, it is the responsibility of the applications running on NOX to decide the route to follow [3].

Such a package is usually the reason for creating a new flow (flow – initiation). Nevetheless, through applications that "run" in NOX, it can be decided to take all the packets corresponding for example to the same protocol, which means that it will never create a new flow. Generally the NOX uses the new import flows and the rest of network traffic that is updated every time the status of the network and to decide whether and which route will promote the movement.

At this time, the NOX is operable to user-space environment in an ordinary PC or Server and can generate 100,000 new flows per second, ie more than enough to handle the traffic of a whole university community. Applications of NOX are generated using either Python's language or in C ++, and loaded dynamically, while the basic infrastructure and functions that are considered critical to the speed have been implemented in C ++.

The programming environment of NOX is basically quite simple, since it revolves around events, network conditions, and assigning "names" to high level and low-level functions. The NOX in the core, provides only low-level methods for interaction with the rest of the network. All high-level methods and events, are created and supported by its applications. In fact, an application is nothing more than a sum of methods [3].

Additionally, however, it has the possibility to define specific methods that can be used by another application. So for example the process of routing of a packet can be implemented in one application and another application that may need this procedure can be declared as a dependency and use it freely. Table X below, showes some of the main applications provided by the NOX. As shown in this table, these applications are basically divided into three main categories according to the network management type considered. These categories are:

- Core apps: They provide a set of functions used by the other two categories of applications for managing the network.
- Network apps: These are applications that manage actually the network and is responsible for taking decisions.
- Web apps: Used by the NOX to provide internet services (Web services) to its users.

Big businesses or academic communities networks are not static and there is a need to create or delete flows, add and remove users, and links whose condition may vary (up / down). An event is something that happens on the network managed by NOX, and which may be of interest to some of its applications. To enable applications NOX face this plethora of events and information, a set of event handlers is used. Various events are associated with their event handlers. Event-handlers are used in accordance with the turn they are declared during NOX startup, and the returned value indicates whether the event-handlers that follow will continue processing the event or not. Some facts are obtained directly through specific OpenFlow messages that NOX can recieve, and other events may be generated by the applications themselves, as a result of specific events such as connection or disconnection of a switch and receiving package or statistics measurements from the switch. Tables 2.7 and 2.8 are shown some of the key events that can be used by applications of NOX. It is worth noting that the NOX administrator has the potential to create additional applications and events, accompanied by their respective event handlers, to handle the NOX network in the desired manner [4].

If we observe the applications of NOX from a more general perspective, it is nothing more than a set of event handlers. Thus these events determine the entire operation of NOX.

Event	**Trigger**
Datapath_join_event	Connect new switch network
Datapath_leave_event	Disconnecting a switch from the network
Packet_in_event	Receive new package from NOX
Flow_mod_event	Adding or modifying a Flow from NOX
Flow_removed_event	Deletion or termination of a flow
Port_status_event	Change the status of a port
Port_stats_in	Receiving a message from Port_stats Controller
Host_event	created by Authenticator each time a new host connects to the network
Flow_in_event	created by Authenticator each time the Controller receives a Packet_in_event
Link_event	created by Discovery with every addition or change of a link (link) to the network

Table 5. Events due to OpenFlow messages received from NOX and events created from NOX applications.

2.2.2 Beacon Controller

Beacon is an OpenFlow controller developed in Java environment and made available for use in 2011. It can run on several platforms (Windows, Linux, Android OS) and supports multithreaded (multithreaded) function. It is based on partial logic (modular) so it is easily expandable using new parts (modules) to support additional functions.

Using code bundles, Beacon can perform dynamic applications that are not interdependent, starting, stopping or pausing bundles code according to the user's options without restarting the controller for each change. This architecture of the Beacon is shown in Figure 5. Code can work together, to share the source packages and command (Java packages) with other bundles, expand the repertoire of their functions with another code (extend) or have multiple versions running at the same time (versioning) [3].

Beacon is supplied with all the essential code packages such as OpenFlow (OF 1.0 Protocol) for OpenFlow protocol packages for encoding and decoding packet (Ethernet, ARP, IPv4, LLDP, TCP, UDP), packages for basic switch functions (Core, Learning Switch, Hub, Device Manager) and sets of algorithms and finding topology (Topology, Layer 2 Shortest Path Routing).

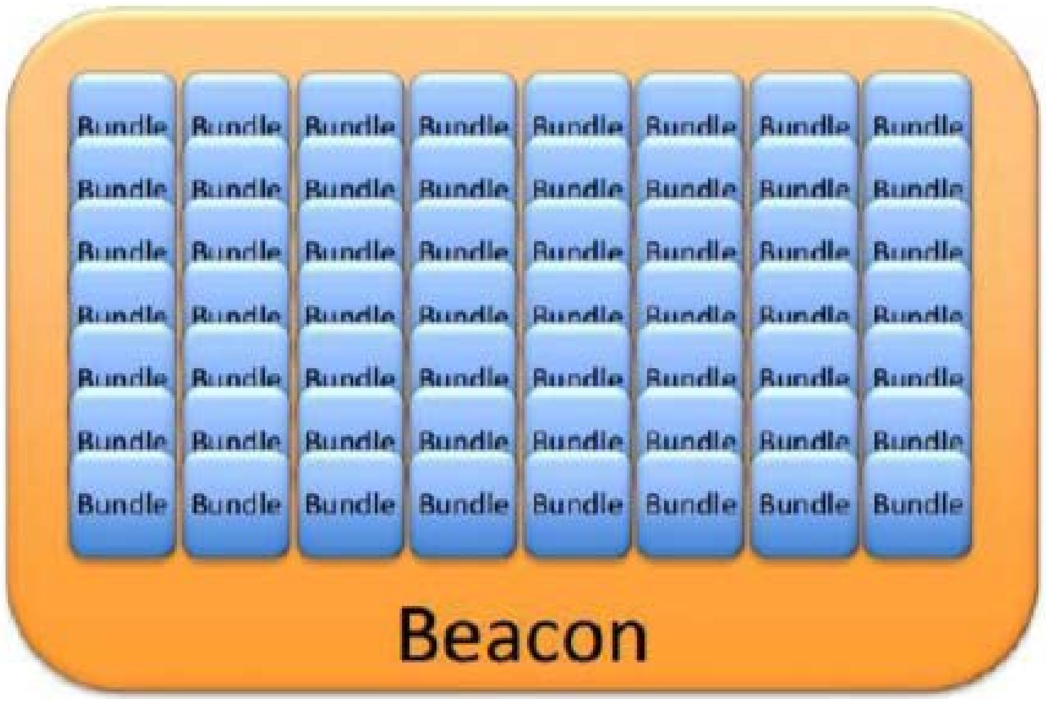


Figure 5. Beacon controller with bundles code.

2.3 Mininet software

The Mininet program is a network emulator. It has the ability to simultaneously perform a set of terminals, routers, ethernet switches and respective links to a single Linux Kernel (core). It uses virtualization technology to enable a single system to be simulated as a complete network using the same core system and the same passwords. Each virtual terminal in mininet works like a real terminal. Moreover it enables secure connection (type SSH) in the terminal, executes any program (provided that it is installed on the Linux system) .The running programs can send packets between the terminals as well as recognizes the link between the interfaces as type Ethernet. While sending of packets is carried out by a given connection speed and the required delay. The packets are processed by devices that operate as routers (Ethernet swtiches, routers) with a given time in queues. When two programs, such as the iperf (which measures the capacity of the line between two points) between a client (client) and a server (server) communicate via Mininet, the measured performance should be shared with that of two native machines [6].

Briefly, in Mininet, terminals, routers, switches, controllers and connections are created using software and not hardware. It is possible to create a Mininet network similar to a real network based on hardware, or the creation of a hardware network similar to that of Mininet, which performs the same binary code and applications on each platform.

As can be seen Mininet is a handy and reliable tool to simulate networks garnering significant advantages. Below the most notable advantages are summarized [6].

- The creation of a simplified network takes place in no time, making it possible to quickly perform the process of debugging.
- The execution of all software supported by the Linux operating system is possible.
- It may modify the packet forwarding: Mininet Routers can be programmed using OpenFlow protocol.
- Mininet can run on any computer, server, virtual machine or even cloud-type technology (cloud computing).

- The results of the software can be played back by any user as all that is required is to run the same code in the corresponding terminal.
- The Mininet is a handy software. To create and perform experiments in programming that require Python language.
- It is an open source project and is under active development. The Mininet community consists of users and developers and can help to address any problem that may be faced by each user.

2.3.1 Topologies in Mininet

Mininet supports the creation of customizable topologies. With the creation of the corresponding Python code, it is possible to create flexible topology which can be configured based on the information included in the code, and can be reused in multiple experiments [6].

For example, the following illustrates a network topology consisting of a specified number of users (hosts) and associated with a switch.

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
class SingleSwitchTopo(Topo):
"Single switch connected to n hosts."
def __init__(self, n=2, **opts):
# Initialize topology and default options
Topo.__init__(self, **opts)switch = self.addSwitch('s1')
# Python's range(N) generates 0..N-1
for h in range(n):
host = self.addHost('h%s' % (h + 1))
self.addLink(host, switch)
def simpleTest():
"Create and test a simple network"
topo = SingleSwitchTopo(n=4)
net = Mininet(topo)
net.start()
print "Dumping host connections"
dumpNodeConnections(net.hosts)
print "Testing network connectivity"
net.pingAll()
net.stop()
if __name__ == '__main__':
# Tell mininet to print useful information
setLogLevel('info')
simpleTest()
```

Classes and functions used are explained further:

- Topo: The base class used in topologies Mininet
    - addSwitch (): adds a router in the topology and returns the name of the router
    - addHost (): adds terminal in topology and returns the name
    - addLink (): adds a two-way connection in topology. (Connections to Mininet both ways unless otherwise stated.)
- Mininet: main class for the creation and management of the network
    - start (): Enables the operation of the network.
    - pingAll (): check the connectivity of the terminal by performing successive ping requests between nodes.
    - stop (): Terminates the network operation. net.hosts: Returns the name of all
    - dumpNodeConnections nodes (): Rejects connections to / from a set of nodes

2.3.2 Setting performance parameters

Besides the basic functions of networking, Mininet provides configurable performance and isolation of certain characteristics, through CPULimitedHost and TCLink classes.

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
class SingleSwitchTopo(Topo):
"Single switch connected to n hosts."
def __init__(self, n=2, **opts):
Topo.__init__(self, **opts)
switch = self.addSwitch('s1')
for h in range(n):
# Each host gets 50%/n of system CPU
host = self.addHost('h%s' % (h + 1),
cpu=.5/n)
# 10 Mbps, 5ms delay, 10% loss, 1000 packet queue
self.addLink(host, switch,
bw=10, delay='5ms', loss=10, max_queue_size=1000, use_htb=True)
def perfTest():
"Create network and run simple performance test"
topo = SingleSwitchTopo(n=4)
net = Mininet(topo=topo,
host=CPULimitedHost, link=TCLink)
net.start()
print "Dumping host connections"
dumpNodeConnections(net.hosts)
print "Testing network connectivity"
net.pingAll()
print "Testing bandwidth between h1 and h4"
h1, h4 = net.get('h1', 'h4')
net.iperf((h1, h4))
net.stop()
if __name__ == '__main__':
setLogLevel('info')
perfTest()
```

The most important methods and parameters used are explained further:

- self.addHost (name, cpu = f): With the use of this command allows the definition of the percentage of total system CPU that will use the virtual user.
- self.addLink (node1, node2, bw = 10, delay = '5ms', max_queue_size = 1000, loss = 10, use_htb = True): Creates a bidirectional connection between two nodes with specific characteristics such as capacity, delay, packet loss tolerance, with a maximum queue size of 100 packets. The parameter bw is expressed in Mb / s, while the delay is followed by the corresponding unit time (s, ms, us) .Antitheta loos the parameter is expressed in percentage.

2.3.3 Run programs in virtual terminals

Running programs in terminals is the most memorable event during the execution of the experiments, so further commands can be supported from the usual pingAll () and iperf () type commands. This process is supported by the Mininet software. Each terminal in Mininet is basically a bash shell type associated with one or more network interfaces thus support running bash type commands. For this reason, to communicate with each terminal is mainly used method type CMD. To execute a command from a host and imprinting effect, through method cmd, the following code is used [6]

```
h1 = net.get('h1')
result = h1.cmd('ifconfig')
print result
```

In many cases the execution of an order in the spotlight for some time is required, stopping or storing the result in a file.

```
from time import sleep
...
print "Starting test..."h1.cmd('while true; do date; sleep 1; done >
/tmp/date.out &')
sleep(10)
print "Stopping test"
h1.cmd('kill %while')
print "Reading output"
f = open('/tmp/date.out')
lineno = 1
for line in f.readlines():
print "%d: %s" % ( lineno, line.strip() )
lineno += 1
f.close()
```

Apart from the use of the shell waiting facility, Mininet enables lets you run a set of commands using the command sendCmd (), and then, which is expected to be completed at a later time using the command waitOutput ():

```
for h in hosts:

h.sendCmd('sleep 20')

…

results = {}

for h in hosts:

results[h.name] = h.waitOutput()
```

2.3.4 Mininet File System

Virtual hosts in Mininet share by default the root folder of the underlying server system. Instead, creation of a new separate system (filesystem) is time consuming and very difficult.

The shared file system provides the advantage that you will not need to copy data between hosts as they have been already created. [5]

This, however, has one major drawback. If special configuration is required for a program (eg. Httpd), creation of new configuration files for each host is required. Furthermore it creates the risk of conflict files, if the same file is created in the same directory.

2.3.5 Configuration methods of hosts

Hosts in mininet provide a number of processes that contribute to the ease of network configuration [6].

- IP (): Returns the IP address of the terminal as a specific interface.
- MAC (): Returns the MAC address of the terminal as a specific interface.
- setARP (): Creates a static ARP entry in the ARP cache of the terminal.
- setIP (): Settings specific IP address for a terminal interface.
- setMAC (): Settings specific IP address for a terminal interface.

For example:

```
print "Host", h1.name, "has IP address", h1.IP(), "and MAC address", h1.MAC()
```

2.3.6 Mininet CLI

The Mininet includes Command Line Interface that can operate on a network. It provides a variety of useful commands, and the ability to display xterm window for execution on individual nodes of a network [5].

```
from mininet.topo import SingleSwitchTopo
from mininet.net import Mininet
from mininet.cli import CLI
net = Mininet(SingleSwitchTopo(2))
net.start()
CLI(net)
net.stop()
```

Using command-line helps to debug the network, displays the network topology (using the command net), check the connectivity (with pingall command) and send commands to all terminals independently.

```
*** Starting CLI:
mininet> net
c0
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (0/2 lost)
mininet> h1 ip link show
746: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
749: h1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP qlen 1000
link/ether d6:13:2d:6f:98:95 brd ff:ff:ff:ff:ff:ff
```

2.3.7 Mininet API

We call API or Application Programming Interface known as Application Programming Interface, the interface of programming procedures that an operating system, library or application provides to allow this to be done requests from other programs and / or data exchange [6].

The previous paragraphs presented a number of classes of Python included the API of Mininet, including headings Topo, Mininet, Host, Switch, Link and their subcategories. Due to simplify and facilitate planning classes are divided into three categories-levels: high-level API, medium-level API and low level API [6].

Low level API: It consists of the main classes relating to nodes and links (as Host, Switch, and Link and its subclasses) are used to create a network. The way of setting up a network with only headings of that level is particularly cumbersome process.

Moderate API: Adds Mininet type objects, which serve as additional information and settings in knots and prudence. Provides a set of methods (such addHost (), addSwitch (), and addLink ()) for the addition of nodes and links in the network, and the network configuration, startup and shutdown (start (), stop ( )).

High level API: Provides additional setting options of topology. Through topo Class offers the possibility to create topology model that can be customized and reused. Models of this type are defined via command mn (through argument - custom option) and executed from the command line.

Generally for the direct control of nodes and routers, the low-level API used. Instead of stopping the operation of a network used the API intermediate level.

The creation of a complete and very detailed network may be accomplished using any standard API, but is usually selected in the mid and high level thanks to the classes that contain greatly facilitate the creation. Here are three examples of creating a complete network using a different API level each time [6].

Low-level API: nodes and links

```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

Mid-level API: Network

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
net.start()
print h1.cmd( 'ping -c1', h2.IP() )
CLI( net )
net.stop()
```

High-level API: Topology example

```
class SingleSwitchTopo( Topo ):
"Single Switch Topology"
def __init__( self, count=1, **params ):
Topo.__init__( self, **params )
hosts = [ self.addHost( 'h%d' % i )
for i in range( 1, count + 1 ) ]
s1 = self.addSwitch( 's1' )
for h in hosts:
self.addLink( h, s1 )
net = Mininet( topo=SingleSwitchTopo( 3 ) )
net.start()
CLI( net )
net.stop()
```

To middle level API consists of simpler structure as shown in the above example requires the creation of class topology. If the low- and medium-level API are flexible, they have the disadvantage of being more cumbersome their reuse as opposed to the high-level API.

2.3.8 Measurement tools

Mininet contains command tools for recording measurements and help control the network and debugging effort. Here are the most important tools along with their respective commands [6].

- Bandwidth (bmw-ng, ethstats)
- Delay (via ping command)
- Queues (through the tc command included in Class monitor.py)
- Statistics TCP (tcp_probe)
- CPU use (top, cpuacct)

3. Traffic Engineering & SDN

The path control process through which the traffic is handled in the network is called Traffic Engineering-TE. There are many reasons why network managers wish to influence the characteristics of a path, one of which is the use of optimization of network resources. The purpose is simple: avoid the situation of certain parts congestion when other are underutilized. Other important reasons are the path to have certain limitations -constraints (eg not to use long delay links), so in line collapse cases to ensure fair priority in the distribution of motion. Through this process of Traffic Engineering new services are offered with extensive Quality of Service guarantees and investments decline in new network resources such as bandwidth, by optimizing the use of existing ones. It has been shown in practice that the technology of MPLS, and by extension the successor of the Generalized, offer the required operational flexibility simultaneously with simplicity to implement complex policies TE [8].

One of the most powerful and useful features of Mininet is that it uses Networks Defined Software. By using the OpenFlow protocol the programming of routers is allowed so that they can make decisions for packets that enter. The OpenFlow technology makes simulators like Mininet more useful since the design of network systems, including custom packet forwarding using OpenFlow, can be transferred to OpenFlow routers functions on the line rate [8].

3.1 OpenFlow and custom routing

In performing an experiment mininet uses default ovsc type controller. The corresponding equivalent command is [9]:

```
$ sudo mn --controller ovsk
```

A Controller of this type implements a simple Ethernet router learning, and supports up to sixteen individual routers. When constructing a script (Script), when the class Mininet () is executed, a corresponding class of the controller should be set and. If a user of a specific controller class is not declared then it is called the default Controller () class thus creating Stanford / OpenFlow type controllers [8].

Conversely, the possibility of use of different type of controller is provided, depending on the needs of the application. The user can create a subclass Controller () and transfer it in the Mininet system files [10].

RemoteController () functions as an intermediary for a controller that can operate anywhere in the control network, except that uptime and downtime should be done in a manual way or with a device that is not controlled by the Mininet. The controller is a function and not an object. It is possible to create a building function in series using the argument partial or lambda or creating a function that takes arguments and returns the controller object. Finally the possibility is given of introducing the controller as class (subclass of class RemoteController ()).

One also useful property is that it is possible to create multiple controllers and to create a subclass of Class Switch () that allows connection to different controllers.

3.2 Floodlight controller

Floodlight Controller is based on Java, as Beacon, but has a different architecture and operating mode. The controller comprises an autonomous collection of modules which perform the main functions of Floodlight as OpenFlow controller and applications are developed so as to overlap (on-top) the base unit controller (REST applications) or to operate together with the controller (Module applications), as shown in the following scheme [6]:

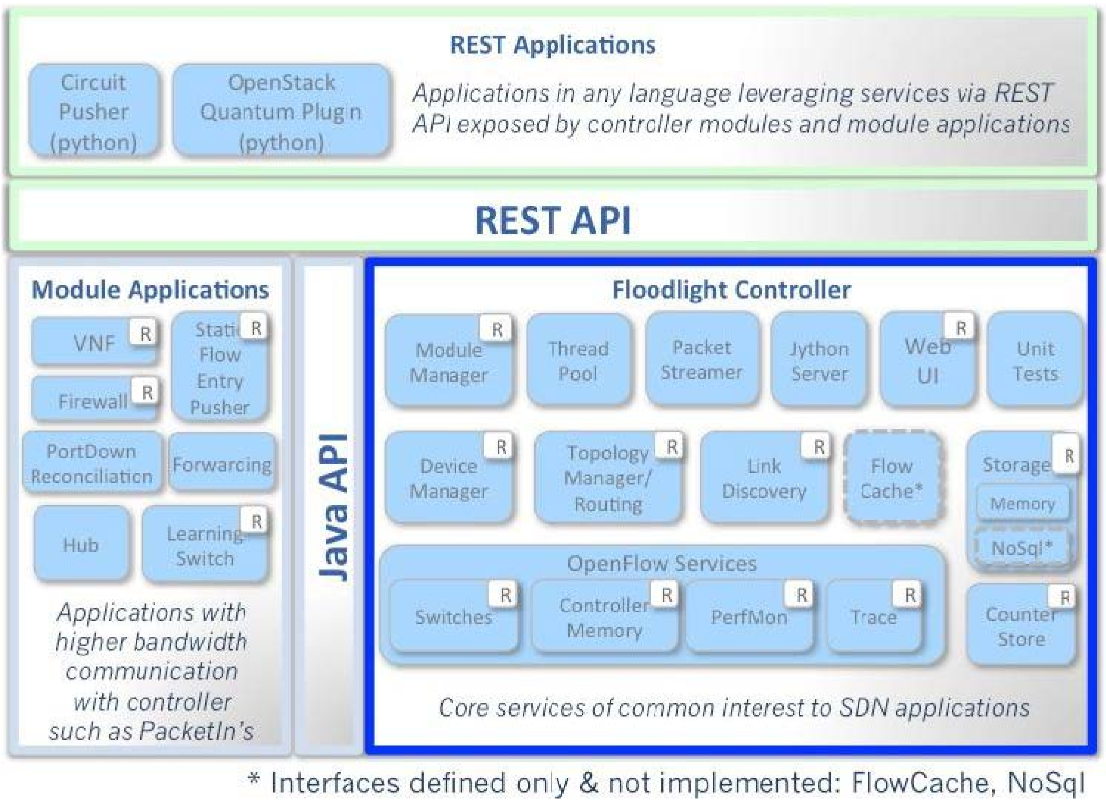


Figure 6. Floodlight controller and REST applications

As shown above, Floodlight consists of functional topology and link modules (Topology Manager, Link Discovery), control devices and units (Device Manager, Module Manager), OpenFlow services (OpenFlow Services), unit storage and handling (Storage, Counter Store, Flow Cache, Packet Streamer, Thread Pool).

Applications that use Floodlight as an underlying layer are developed as independent Java modules and use the programming REST interface of the controller (REST API -

application programming interface). Through its REST API applications Floodlight communicates with the controller and can use the network for any function [8].
Also applications can be developed and adjusted to the level of Floodlight controller, enough to develop and implement the controller modules as Java (Java modules). This way, although more demanding and difficult than the use of the REST API's, has significant benefits as regards the execution speed of the application and greater communication bandwidth offer with Floodlight, since the coupling of application - controller becomes more directly to the use of Java API's.

3.3 Topology example

Mininet API allows the users to create custom networks depending on their needs, using a few lines of code in Python. In this section we will present a topology example that is presented in the Mininet manual [6].

Custom topology example
Two directly connected switches plus a host for each switch:
  host --- switch --- switch --- host
Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.

```
from mininet.topo import Topo
class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )


        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

According to this code above we see that our topology consists of three terminals and four routers to the corresponding connections shown. The code includes the topology and the elements that constitute it. In this way one can create the topology that meets the needs of the application to be able to perform the simulations. Then to perform the simulation we need to go back to mininet using the following command.

```
$ sudo mn -custom topology.py –topo mytopo – mac-controller=remote,
ip=[controller IP],port=[controller listening port]
```

At this point we will illustrate the above commands in order to understand the use of each command. By sudo mn, the user gets administrator rights on mininet and instruct the mininet to perform topology.py file located in the custom folder and create mytopo topology with the corresponding links to govern as defined in file topology. py. The reason for this label from the program is that the file may include more than one topologies. In addition, the argument -mac sets the MAC address of each terminal equal to the IP address. This is done for simplicity in the process of analyzing the results. In addition, for creating traffic from terminal to terminal and general control functions of terminals, we use argument -x which has resulted in the creation of execution window (command prompt), one for each device. Finally the most important argument is the -controller = remote, ip = []. With this command define the topology controller (OpenFlow Controller) that is located in the ip address []. This Controller can be a Floodlight controller. This fact allows the controller to be located somewhere remotely as in this case where it is outside the virtual machine [6].

4. SDN/Traffic Engineering based on Kruskal

4.1 Kruskal algorithm

4.1.1 Dynamic programming

The term dynamic programming was introduced in 1953 by Richard Bellman in order to describe the process of solving problems that are broken down into a sequence of consecutive decisions.

It is a method that is applicable when sub problems are not independent. An algorithm is a product of dynamic programming that solves each sub problem once and store this solution in a table, in which the algorithm will resort every time this problem is met. This is a very powerful technique for solving algorithmic problems [9].

Dynamic programming is applied generally to optimization problems. In these problems it can give many different solutions.

Each solution has a specific value, and the aim is to find the solution with the optimal (minimum or maximum) value. A solution that ensures optimum value is designated as an optimal solution, since there may be different options that achieve the optimal value [10].

The development of a dynamic programming algorithm can be decomposed into a series of four steps.

1. We characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution working "bottom-up" or "by some to general."

4. We construct an optimal solution from the data we have calculated.

Steps 1-3 are the basic stages of solving a problem with the method of potential programming. Step 4 may be omitted if the aim is just the value of an optimal solution.

4.1.2 Defining the problem

The "greedy" algorithms are useful tools for optimization solving. For example, the problem of finding the shortest path between two vertices of a graph or finding the optimum range to perform a set of works from a computer system [9].

Problem:
Let G = <N, A> a coherent non-directed graph, where N is the set of vertices and A the set of edges, in each of which a number is assigned, its weight. A subset T of the edges of G must be found, which connects all the vertices of the graph G and the sum of the weights of T can be minimized [11].

T is called minimal overlapping tree (minimum spanning tree / MST). Overlapping is the tree that contains all the nodes of the graph, and minimum is the one containing the minimal edges on all nodes.

4.1.3 Kruskal algorithm description

A greedy algorithm has a simple structure consisting of the following elements [9]:

- a set of candidate choices (eg the top of a graph)
- a set of options that have already been used in the process of solving.
- a control function, which controls whether a particular set of candidate options yields a solution. (not necessarily the best for the time considered)

- a function that checks whether a set of candidate choices possible, to give us a solution to the problem.
- a selection function, which at all times shows what option is the best prospect to be part of the solution.
- an objective function that gives the value of the solution. The objective function is desired to be optimized. (The selection function is typically based on the objective function, and indeed may also be the same).

One such algorithm proceeds in the next step with the decision that now seems to be the best for problem solving. This does not mean that always delivers the optimum solution.

Kruskal algorithm is described by the following basic principles [9]:

- The set T of the edges is initially empty. With the progress of the algorithm, new edges are added to T one by one.
- At any time graphs formed by the vertices of G and edges of T may represent more than one different connected trees. The edges of T contained in each of them are poorly connected trees for all the vertices of each segment. At the end of the algorithm we take only one bonded portion so T is the minimum overlapping tree by G.
- To create larger and larger independent affiliated sections, we look at the edges of the graph G in ascending weight order. If an edge is joining two vertices from different departments, then it is inserted in T and the two parts are now a single connected part. Else, it is rejected because its inclusion in T would create a cycle.

4.1.4 Kruskal theoretical examples

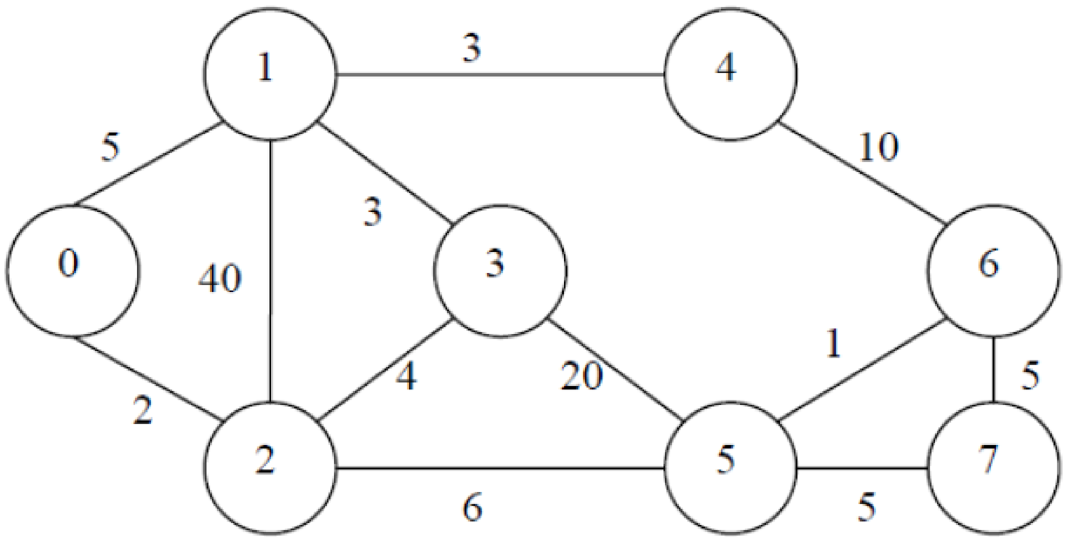


Figure 7. Undirected busy coherent graph for Kruskal example

Consider the above undirected busy coherent graph G = (N, A). In order to show the minimum overlapping tree (Minimum Spanning Tree-MST) that is obtained by applying Kruskal algorithm. Where there is more than one option, the edge joining vertices with the smallest sum of the weights is selected [11].

- Step 1: Add the edge (5,6) having a weight of 1
- Step 2: Add the edge (0.2) having a weight of 2
- Step 3: Add the edge (1,3) having a weight of 3
- Step 4: Add the edge (1.4) having a weight of 3
- Step 5: Add the edge (2,3) having a weight of 4
- Step 6: Acne (0,1) can not be added because it creates cycle
- Step 7: Add the edge (5,7) having a weight of 5
- 8th step: The edge (6,7) can not be added because it creates cycle
- 9th step: Add the edge (2,5) having a weight of 6

So we have the Minimum Spanning Tree-MST, after applying Kruskal algorithm:

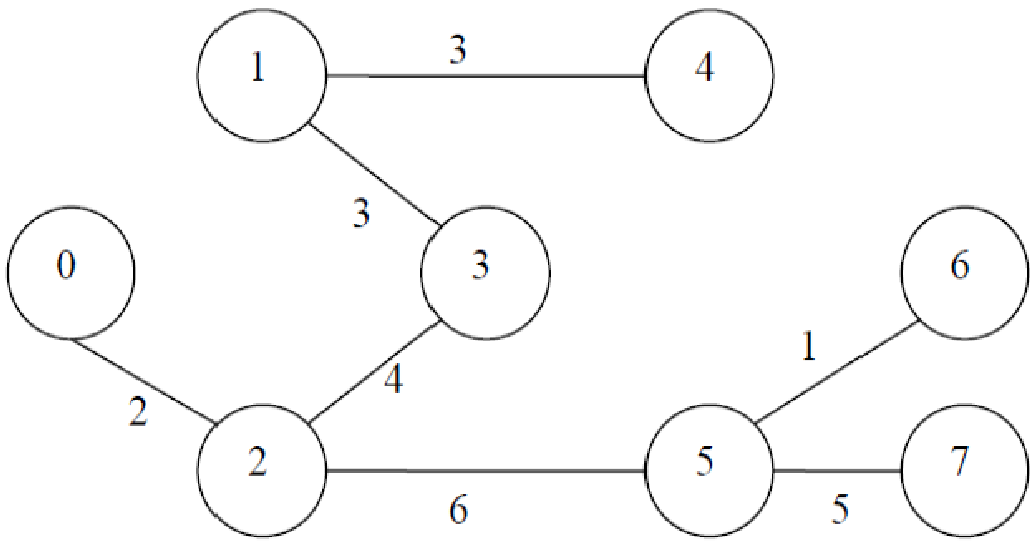


Figure 8. Minimum spanning tree after Kruskal application.

The following code implements the Minimum Spanning Tree with Kruskal algorithm in C++ [9]:

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=0;
int min,mincost=0,c[9][9],parent[9];
int find(int);
int uni(int,int);
int main()
{
printf("Kruskal Algorithm\n");
printf("\nDwste ton arithmo twn komvwn\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
c[i][j]=0;
}
}
//eisagwgh varwn
c[0][1]=0;
c[0][2]=2;
c[1][0]=0;
.
.
```

```
c[7][6]=5;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(c[i][j]==0)
c[i][j]=999;
}
}
printf("\n\nOi koryfes tou MST einai :\n");
while(ne<=n)
{
for(i=0,min=999;i<n;i++)
{
for(j=0;j<n;j++)
{
if(c[i][j]<min)
{
min=c[i][j];
a=u=i;
b=v=j;
}
}
}
u=find(u);
v=find(v);
if(uni(u,v))
{
printf("\n%d edge (%d,%d) =%d\n",ne,a,b,min);
mincost +=min;
}
c[a][b]=c[b][a]=999;
ne++;
}
printf("\n Elaxisto kostos = %d\n",mincost);
system("PAUSE");
}
int find(int i)
{
while(parent[i])
i=parent[i];
return i;
}
int uni(int i,int j)
{
if(i!=j)
{
if(i<j)
parent[i]=j;
else if (i>j)
parent[j]=i;
return 1;
}
return 0;
}
```

The results from running the above algorithm are the following:

For the graph of the theoretical example:

Number of vertices: 8

The tops of the MST are:

- 0 edge (5,6) = 1
- 1 edge (0,2) = 2
- 2 edge (1,3) = 3
- 3 edge (1,4) = 3
- 4 edge (2,3) = 4
- 5 edge (5,7) = 5 7 edge (2,5) = 6

Minimum Cost = 24

The results for application of the same algorithm for the graph of Figure 9 is shown in the following.

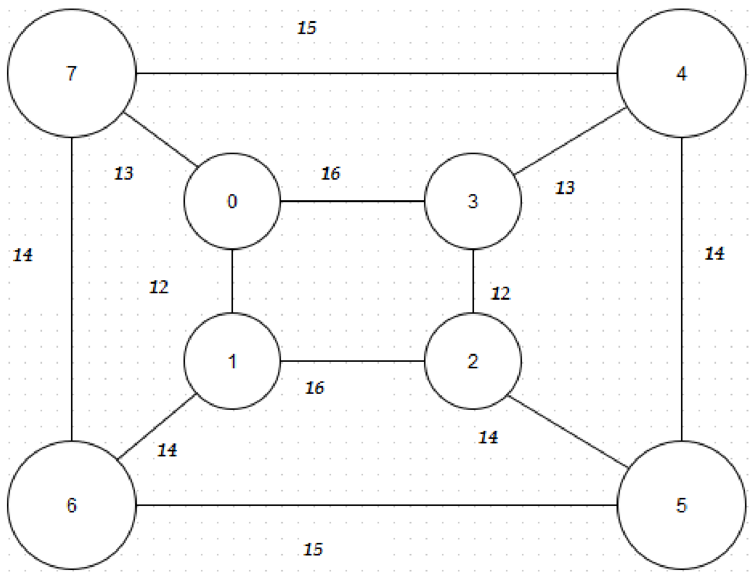


Figure 9. Graph for application of Minimum spanning tree -1.

Number of vertices: 8

The tops of the MST are:

- 0 edge (0,1) = 12
- 1 edge (2,3) = 12
- 2 edge (0,7) = 13
- 3 edge (3,4) = 13
- 4 edge (1,6) = 14
- 5 edge (2,5) = 14 6 edge (4,7) = 15

Minimum Cost = 93

The results for application of the same algorithm for the graph of Figure 9 is shown in the following.

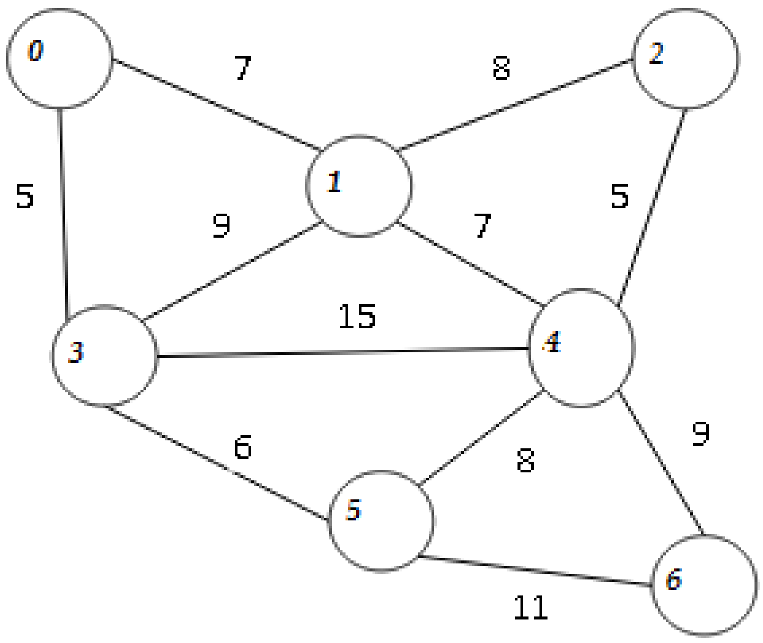


Figure 10. Graph for application of Minimum spanning tree -2.

Number of vertices: 7

The tops of the MST are:

- 0 edge (0,3) = 5
- 1 edge (2,4) = 5
- 2 edge (3,5) = 6
- 3 edge (0,1) = 7
- 4 edge (1,4) = 7
- 5 edge (4,6) = 9

Minimum Cost = 39

4.1.5 Comparison of Kruskal with other algorithms

Kruskal algorithm selects edges, just trying to avoid creating cycles. This mode has the effect of creating a forest of trees, which then grow in an irregular manner, depending on the evolution of the algorithm [10].

On the other hand, Prim algorithm gradually develops the minimum connected tree itself starting from a randomly chosen root. In each step, Prim adds a new branch of the tree and the algorithm is complete when you have selected all the vertices of the graph.

The algorithm of Dijkstra, finds the minimal path between two peaks while Kruskal returns the minimum spanning tree that connects all vertices of a graph.

The algorithm of Bellman-Ford, solves the problem of its baseline light paths in the general case, where the edges may have negative weights. If there are negative weights, the algorithm indicates that the problem has no solution. If not, it calculates the lighter paths and their weights [10].

The Boruvka algorithm, is a most demanding algorithm for finding the minimum spanning tree in a graph in which all edges have different weights.

4.1.6 Complexity

The main loop of the Prim algorithm is performed n-1 times and in each run the loop for enclosing has complexity O (|V|) . Hence, the algorithm has overall complexity O (| V |^2) [11].

Compared with Kruskal algorithm with complexity O (|E| log |V|), (where E: number of edges and V: number of peaks) [11]:

- For "dense" graphs where the number of edges tends to |V| (|V|-1) / 2 Kruskal algorithm has complexity O (|V| 2log |V|) and therefore the Prim algorithm is better.
- For "dilute" graphs where the number of edges tends to V, the algorithm of Kruskal [O (| V | log | V |)] is the most efficient.

The complexity of the Dijkstra algorithm, is O ((|V| + |E|) log |V|) whereas for sparse graphs, complexity is reduced to O (|E| log |V|).

The Boruvka algorithm complexity is O (|E| log |V|).

4.2 Traffic engineering with Kruskal on Floodlight

In this thesis, we implemented the minimum spanning tree algorithm in a custom module of Floodlight controller. The Floodlight controller implements the Openflow protocol. This application aims to building loop topologies by utilization of LearningSwitch and non-STP enabled OF switches, avoiding broadcast storm. This implementation of Floodligth module applies the minimum spanning tree over a network created by Mininet software. The application is implemented in Java.

In this section we will present the implementation of Kruskal and minimum spanning tree in Java. The following code initiates the Kruskal algorithm, comparing the cost of link pairs.

```
if
(reverse)
{
           Collections.sort(topoEdges, new
           Comparator<LinkWithCost>() {
           public int compare(LinkWithCost link1,
           LinkWithCost link2) {
           return new
           Integer(link2.getCost()).compareTo(link1.get
           Cost());
           }
           });
           } else {
           Collections.sort(topoEdges, new
           Comparator<LinkWithCost>() {
           public int compare(LinkWithCost link1,
           LinkWithCost link2) {
           return new
           Integer(link1.getCost()).compareTo(link2.get
           Cost());
           }
           });
```

The following code generates the nodes hashmap, with one entry for each switch. Here, a set of connect components is created for each node.

```
if
(!nodes.containsKey(lt.getS
rc())) {
                            nodes.put(lt.getSrc(), new
                            HashSet<Long>());
                            nodes.get(lt.getSrc()).add(lt.g
                            etSrc());
                            }
                            if
                            (!nodes.containsKey(lt.getDst()
                            )) {
                            nodes.put(lt.getDst(), new
                            HashSet<Long>());
                            nodes.get(lt.getDst()).add(lt.g
                            etDst());
                            }
```

After the execution of the above code, the edges and node structure has been generated, on which Kruskal will be implemented. Next, we enter the Kruskal cycle.

Here, the algorithm checks if the edge has already been computed, in order not to compute the same edge twice.

```
if
(edgesDone.contains(cur
Edge)) {
                        logger.trace("Edge already computed by Kruskal.
                        Not computing again!");
                        } else {
                        edgesDone.add(curEdge);
```

Next the size of source and destination edges are compared. If the size of destination edge is smaller than source edge, then destination edge along with all nodes have to be transferred. Else, source edge and nodes are transferred.

```
if (nodes.get(curEdge.getSrc()).size() >
nodes.get(curEdge.getDst()).size()) {
                                           src =
                                           nodes.get(curEdge.getDst());
                                           dst =
                                           nodes.get(dstHashSetIndex =
                                           curEdge.getSrc());
```

Then all nodes are moved from source set to distance set and the index in nodes array is updated. The Kruskal cycle is ended and the minimum spanning tree is printed.

5. Minimum Spanning Tree in Floodlight

The it..msttraffic package that was implemented in this work includes the following classes: IMSTTrafficService.java, TopologyCostsLoader.java and MSTTraffic.java.

Class TopologyCostsLoader.java is used to make import of type TopologyCosts from the package it..msttraffic.types, and creates a new instance of this type. Class IMSTTrafficService.java creates the homonymous interface which extends the IfloodlightService.

```
public interface IMSTTrafficService extends IFloodlightService
```

The additional elements of the interface that created two essential methods, the getCosts and setCosts which retrieve and set the cost of each route respectively.

```
public TopologyCosts getCosts();

public void setCosts(TopologyCosts costs);
```

We also have three new Sets within which we put the edges of the Minimum Spanning Tree, the edges of our topology and edges for removal. These information are gathered from the respective methods getMSTEdges, getTopoEdges and getReduntantEdges of class LinkWithCost which will be explained below.

```
public Set<LinkWithCost> getMSTEdges();

public Set<LinkWithCost> getTopoEdges();

public Set<LinkWithCost> getRedundantEdges();
```

The MSTTraffic.java class is the main class of our program.

The it..msttraffic.algorithms package contains classes that implement the algorithm used for traffic engineering.

The IminimumSpanningTreeAlgorithm.java is an interface containing the vector with the costs and edges of topology, as defined by LinkWithCost class.

The KruskalAlgorithm.java is the class containing the Kruskal algorithm for calculating the minimum coherent tree for our traffic engineering.

```
protected static Logger logger= LoggerFactory.getLogger(KruskalAlgorithm.class);
```

Here we define a logger where you stored any messages from the class.

Then we have the code

```
public   Vector<LinkWithCost>   perform(List<LinkWithCost>   topoEdges,
boolean reverse) throws Exception {
            Logger.debug("Starting to perform Kruskal algorithm...");

            if (reverse) {
                  Collections.sort(topoEdges, new
Comparator<LinkWithCost>() {
                        @Override
                        public int compare(LinkWithCost link1,
LinkWithCost link2) {
                              return new
Integer(link2.getCost()).compareTo(link1.getCost());
                        }
                  });
            } else {
                  Collections.sort(topoEdges, new
Comparator<LinkWithCost>() {
                        @Override
public int compare(LinkWithCost link1, LinkWithCost link2) {
return new Integer(link1.getCost()).compareTo(link2.getCost());
                        }
                  });

            }
```

where Kruskal algorithm is performed, to vector with the edges of our topology.

```
HashMap<Long, HashSet<Long>> nodes = new HashMap<Long, HashSet<Long>>();


        for (LinkWithCost lt: topoEdges) {
            if (!nodes.containsKey(lt.getSrc())) {

                nodes.put(lt.getSrc(), new HashSet<Long>());
                nodes.get(lt.getSrc()).add(lt.getSrc());
            }

            if (!nodes.containsKey(lt.getDst())) {

                nodes.put(lt.getDst(), new HashSet<Long>());
                nodes.get(lt.getDst()).add(lt.getDst());
            }
        }
```

Here we create a HashMap which will contain our switch, and a set of data with which joined each switch.

```
logger.trace("Kruskal generated the following nodes structure: " +
printNodes(nodes));
```

Here the structure of the nodes is printed on logger as derived from Kruskal algorithm.

```
Vector<LinkWithCost> mstEdges = new Vector<LinkWithCost>();
      Vector<LinkWithCost> edgesDone = new Vector<LinkWithCost>();

      logger.trace("Entering Kruskal cycle...");
      for (LinkWithCost curEdge: topoEdges) {
            logger.trace("curEdge = {}", new Object[] { curEdge });

              if (edgesDone.contains(curEdge)) {
                  logger.trace("Edge already computed by Kruskal. Not
computing again!");
              } else {
                  edgesDone.add(curEdge);
                  if
(nodes.get(curEdge.getSrc()).equals(nodes.get(curEdge.getDst()))) {
                        logger.trace("Edge has source set equal to
destination set. Not considering for MST!");
                  } else {
                        HashSet<Long> src = null, dst = null;
                        Long dstHashSetIndex = 0L;

                        logger.trace("Comparing size of source and
destination of curEdge: (src = {}, dst = {}).", new Object[]
{nodes.get(curEdge.getSrc()).size(),
nodes.get(curEdge.getDst()).size()});
                        if (nodes.get(curEdge.getSrc()).size() >
nodes.get(curEdge.getDst()).size()) {

                              src = nodes.get(curEdge.getDst());
                              dst = nodes.get(dstHashSetIndex =
curEdge.getSrc());
                        } else {

                              src = nodes.get(curEdge.getSrc());
```

```
                                    dst = nodes.get(dstHashSetIndex = 
curEdge.getDst());
                                }
                                logger.trace("Set src = {}, dst = {}.", new 
Object[] {printHash(src), printHash(dst)});

                                Object[] srcArray = src.toArray();
                                int transferSize = srcArray.length;

                                logger.trace("Moving each node from set: src 
into set: dst.");
                                logger.trace("Updating appropriate index in 
array: nodes.");
                                for (int j = 0; j < transferSize; j++) {
                                        if (src.remove(srcArray[j])) {
                                                dst.add((Long) srcArray[j]);
                                                nodes.put((Long) srcArray[j], 
nodes.get(dstHashSetIndex));
                                        } else {
                                                logger.error("Error while removing 
element {} from array {}.", new Object[] {srcArray[j], src});
                                                throw new Exception("Kruskal - 
Error performing Kruskal algorithm (set union).");
                                        }
                                }
                                logger.trace("Kruskal updated the nodes 
structure: " + printNodes(nodes));

                                logger.trace("Kruskal add the edge {} to 
mstEdges.", new Object[] {curEdge});
                                mstEdges.add(curEdge);
                        }
                }
        }
        logger.trace("End of Kruskal cycle.");
        logger.debug("Computed MST by Kruskal: "  + printEdges(mstEdges));

        logger.debug("End of Kruskal algorithm.");
```

The above code includes the steps for implementing the Kruskal algorithm, at each stage of which we have detailed messages to the logger.

```
private static String printEdges(Iterable<LinkWithCost> edges) {
      String s  = "\n";
      for (LinkWithCost e: edges) {
            s += e.toString() + "\n";
      }
      return s;
    }
```

This method prints the edges raised.

```
private static String printNodes(HashMap<Long, HashSet<Long>> nodes) {
            String s  = "\n";
      for (Map.Entry<Long, HashSet<Long>> entry: nodes.entrySet()) {
            s += "Node (" + entry.getKey() + "): " +
printHash(entry.getValue()) + "\n";
      }
      return s;

      }
```

This method prints the nodes encountered.

```
private static String printHash(HashSet<Long> value) {
            String s  = "(";
            for (Long set : value) {
                  s += set + ", ";
            }
            s += ")";
      return s;
      }
```

This method prints the HashMap containing switches and links.

The it..msttraffic.types package contains two classes that define the network topology and the cost of each move from node to node.

The LinkWithCost.java is a class that extends the known class Link and contains the network topology. Accepts variables for the ID and Port of the original element, the ID and Port of destination and root cost.

```
TopologyCosts costs = TopologyCostsLoader.getTopologyCosts();
```

The costs of the topology are stored in a TopologyCosts type variable defined in another class package.

```
public int getCost() {
	TopologyCosts costs = TopologyCostsLoader.getTopologyCosts();
	return costs.getCost(this.getSrc(), this.getDst());
}
```

This function draws the costs of getTopologyCosts method Class TopologyCosts, which we will see later.

```
public String toString() {
	return "Link (" + this.getSrc() + ", " + this.getDst() + ") with
cost: " + this.getCost();
	}
```

Printing the results, ie the source, the destination and cost.

```
public LinkWithCost getInverse() {
	return new LinkWithCost(this.getDst(), this.getDstPort(),
this.getSrc(), this.getSrcPort());

	}
```

Reverse destination and source.

The it..msttraffic.web package includes five classes responsible for the elaboration of the topology of the network.

The MSTTrafficEdgesResource.java extends the known ServerResource and runs a IMSTTrafficService to add to existing links, some new.

The MSTTrafficWebRoutable assumes data linking with a Router so that they are available on the network and remote users using the JSON and Restlet (http://restlet.com, http://json.org/)

```
router.attach("/topocosts/json", TopoCostsResource.class);
router.attach("/mstedges/json", MSTTrafficEdgesResource.class);
router.attach("/topoedges/json", TopoEdgesResource.class);
router.attach("/redundantedges/json", RedundantEdgesResource.class);
return router;
```

In this way, we connect the router variable to the cost of the topology, the topology edges and edges that have been removed, using the relevant classes of the package TopoCostsResource, MSTTrafficEdgesResource, TopoEdgesResource, RedundantEdgesResource.

The it..msttraffic.web.serializers package contains classes that make data serialization. This means converting the objects into a series of bytes, which contain information and Object type. The LinkWithCostJSONSerializer.java and TopologyCostsJSONSerializer.java contain classes for cost topology and costs of the edges serialization, while we have the corresponding class for desirialization cost topology, TopologyCostsJSONDeserializer.java.

The serialization is provided by a serializer of packet JSON.

```
 jGen.writeStringField("sourceSwitch",
HexString.toHexString(link.getSrc()));
```

```
        jGen.writeNumberField("sourcePort", link.getSrcPort());
        jGen.writeStringField("destinationSwitch",
HexString.toHexString(link.getDst()));
        jGen.writeNumberField("destinationPort", link.getDstPort());
        jGen.writeNumberField("cost", link.getCost());
```

We see here the serialization of the source, destination and costs.

```
jGen.writeStartObject();

        HashMap<String, Integer> prop = costs.getCosts();
        for (Entry<String, Integer> curProp : prop.entrySet()) {
            jGen.writeNumberField(curProp.getKey(),
curProp.getValue());
        }

        jGen.writeEndObject();
    }
```

Here, we have the serialization of the whole HashMap that contains the switches and connections.

Having seen the individual classes, we better understand the main class of our program, ie MSTTraffic. Here, we define variables necessary for service.

```
protected HashSet<LinkWithCost> topoEdges = new HashSet<LinkWithCost>();
protected HashSet<LinkWithCost> redundantEdges = new
HashSet<LinkWithCost>();

private IMinimumSpanningTreeAlgorithm algorithm = new
KruskalAlgorithm();
```

Definition of data structures to store results.

The method

```
protected void updateLinks()
```

takes on to "listen" to change of topology according to the edges that were added or removed. Uses LinkWithCost.I method that implements changes in topology, removing edges were excluded by Kruskal.

This method saves the changes in a HashSet.

```
protected HashSet<LinkWithCost> findRedundantEdges(Vector<LinkWithCost>
mstEdges)
```

The method

```
protected void modPort(DatapathId datapathId, OFPort ofPort, boolean
open)
```

 finds the ports for hardware address identification and ID of switches.

```
protected String printEdges(Iterable<LinkWithCost> edges) {
      String s  = "";
      for (LinkWithCost e: edges) {
            if (!s.equals("")) s += "\n";
            s += e.toString();
      }
      return s;
      }
```

Here we print all edges.

Next we define necessary classes since we extended interface from source code of floodlight.

```
public void startUp(FloodlightModuleContext context) {
            if (topology != null) topology.addListener(this);
            if (restApi != null) restApi.addRestletRoutable(new
MSTTrafficWebRoutable());
      }

      @Override
      public Set<LinkWithCost> getTopoEdges() {
            return topoEdges;
      }

      protected void setTopoEdges(HashSet<LinkWithCost> topoEdges) {
            this.topoEdges = topoEdges;
      }

      @SuppressWarnings("unchecked")
      @Override
    public Set<LinkWithCost> getMSTEdges(){
            HashSet<LinkWithCost> mstEdges = (HashSet<LinkWithCost>)
topoEdges.clone();
            mstEdges.removeAll(redundantEdges);
      return mstEdges;
    }

      @Override
    public Set<LinkWithCost> getRedundantEdges(){
      return redundantEdges;
    }

      @Override
      public TopologyCosts getCosts() {
            return TopologyCostsLoader.getTopologyCosts();
      }

      @Override
      public void setCosts(TopologyCosts newCosts) {
            TopologyCosts costs = getCosts();
            //costs.getCosts().clear();
            costs.getCosts().putAll(newCosts.getCosts());

            updateLinks();
        }
```

Finally here we have the startup classes and init that are necessary for startup and initialization of the program. Methods setTopoEdges, getMSTEdges, getRedundantEdges, getCsosts and setCosts are defined here but their corresponding classes are in separate files and their function is described above.

5.1 System Analysis

In this session we will use the floodlight controller to create a traffic engineering application.

The project MSTTraffic calculate the minimum spanning tree of a network of four switches. To do this we use the kruskal algorithm which calculate the cost of the edges and links (more details we can find in java code).

5.2 Installation of system

In this session we can find the step-by-step tutorial to install and run the project MSTTraffic.

Step 1

Before installing the floodlight controller, we have to install the JDK and Ant. We write to the console as follows:

```
$sudo apt-get install build-essential default-jdk ant python-dev eclipse
```

Step 2

Now we have to install floodlight controller with the following commands (you need to install git first)

```
$ git clone git://github.com/floodlight/floodlight.git

$ cd floodlight

$ ant

$ sudo mkdir /var/lib/floodlight

$ sudo chmod 777 /var/lib/floodlight
```

Step 3

With the following command we install mininet

```
$ sudo apt-get install mininet
```

Step 4

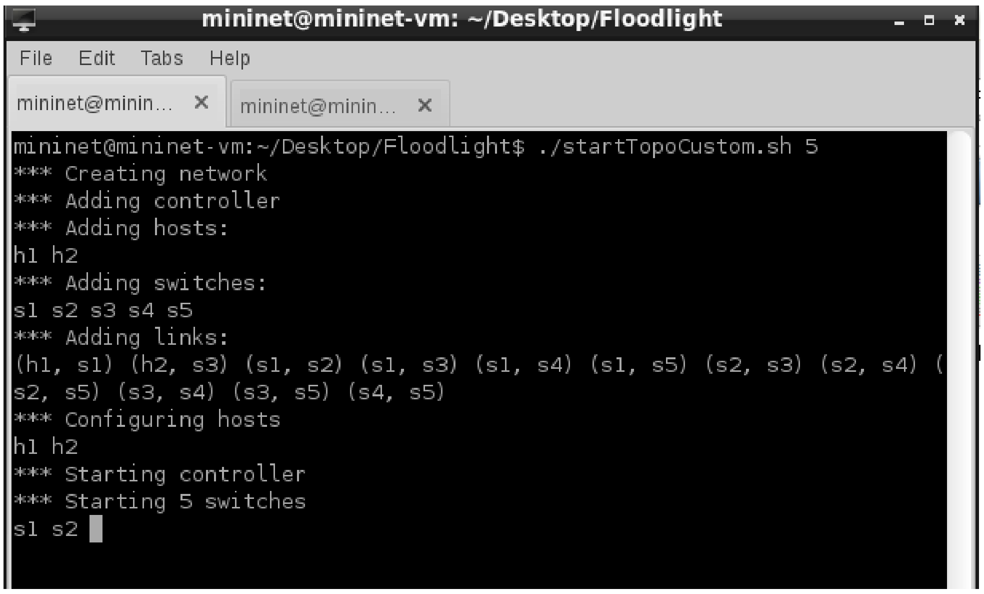
The following action integrate the floodlight controller to eclipse

```
$ ant eclipse
```

6. Execution Test & Results

The package described in Chapter 5 has been implemented in various Mininet network topologies. In this Chapter, step-by-step screenshots will be presented relative to the execution of minimum spanning tree package on topologies with 5 and 10 switches.

Initially we start the Floodlight controller and then we define the custom Mininet topology with 5 switches.



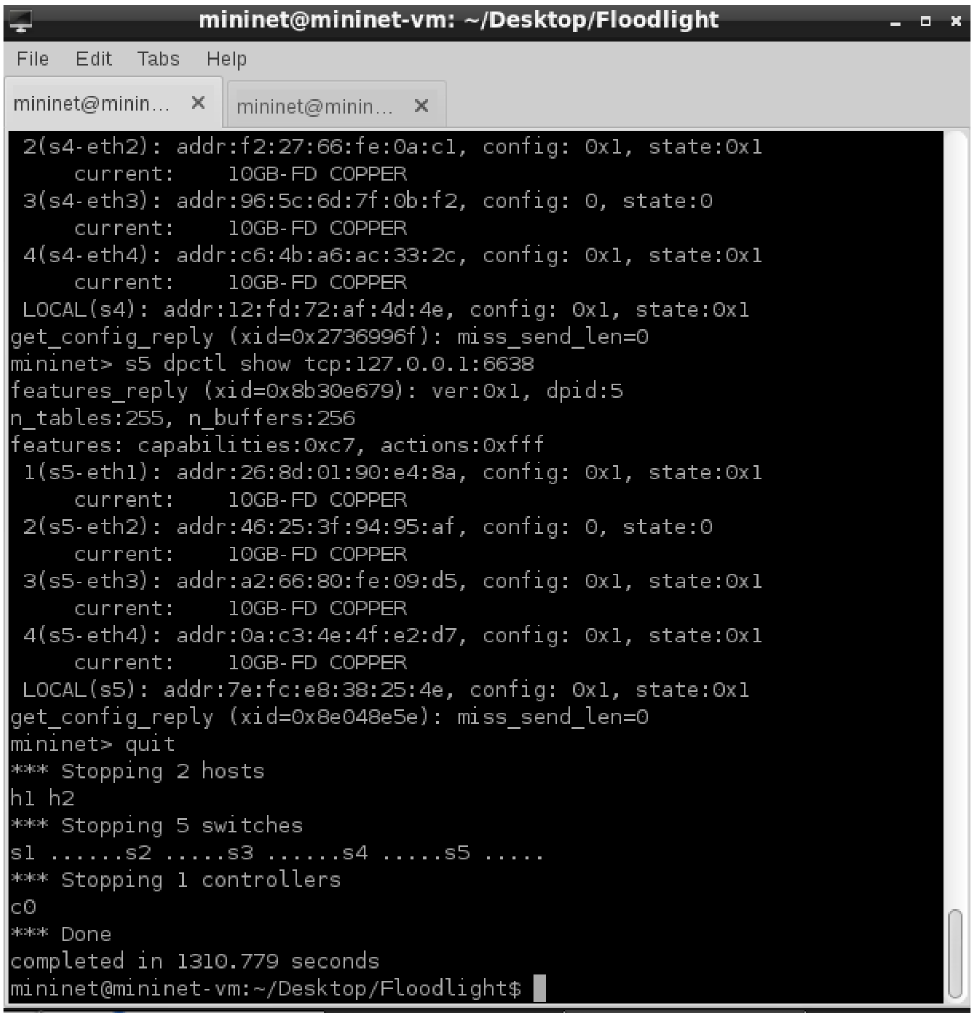
```
mininet@mininet-vm:~/Desktop/Floodlight$ ./startTopoCustom.sh 5
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s3) (s1, s2) (s1, s3) (s1, s4) (s1, s5) (s2, s3) (s2, s4) (
s2, s5) (s3, s4) (s3, s5) (s4, s5)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 5 switches
s1 s2
```

Figure 11. Start Mininet topology with 5 switches

In order to create a Mininet Topology of 10 switches we have to quit the 5-switch topology and then start the 10-switch topology.

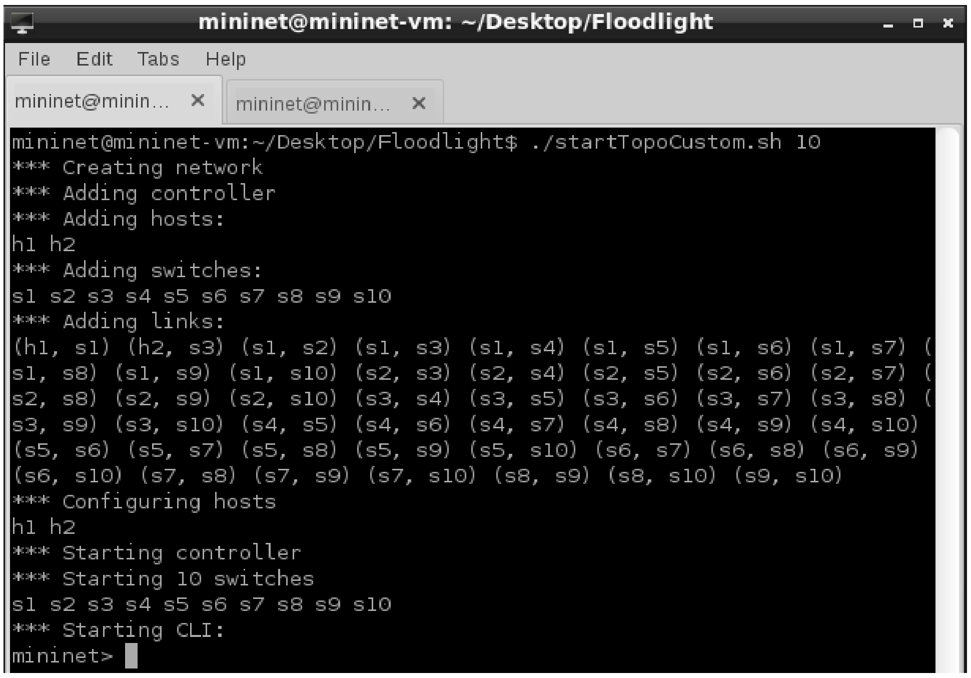
The topology can be shown with the following command:

```
$ ./viewMSTapis.sh -a topoedges
```



```
 2(s4-eth2): addr:f2:27:66:fe:0a:c1, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 3(s4-eth3): addr:96:5c:6d:7f:0b:f2, config: 0, state:0
     current:    10GB-FD COPPER
 4(s4-eth4): addr:c6:4b:a6:ac:33:2c, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 LOCAL(s4): addr:12:fd:72:af:4d:4e, config: 0x1, state:0x1
get_config_reply (xid=0x2736996f): miss_send_len=0
mininet> s5 dpctl show tcp:127.0.0.1:6638
features_reply (xid=0x8b30e679): ver:0x1, dpid:5
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
 1(s5-eth1): addr:26:8d:01:90:e4:8a, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 2(s5-eth2): addr:46:25:3f:94:95:af, config: 0, state:0
     current:    10GB-FD COPPER
 3(s5-eth3): addr:a2:66:80:fe:09:d5, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 4(s5-eth4): addr:0a:c3:4e:4f:e2:d7, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 LOCAL(s5): addr:7e:fc:e8:38:25:4e, config: 0x1, state:0x1
get_config_reply (xid=0x8e048e5e): miss_send_len=0
mininet> quit
*** Stopping 2 hosts
h1 h2
*** Stopping 5 switches
s1 ......s2 .....s3 ......s4 .....s5 .....
*** Stopping 1 controllers
c0
*** Done
completed in 1310.779 seconds
mininet@mininet-vm:~/Desktop/Floodlight$ 
```

Figure 12. Quit Mininet topology with 5 switches



```
mininet@mininet-vm:~/Desktop/Floodlight$ ./startTopoCustom.sh 10
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(h1, s1) (h2, s3) (s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (
s1, s8) (s1, s9) (s1, s10) (s2, s3) (s2, s4) (s2, s5) (s2, s6) (s2, s7) (
s2, s8) (s2, s9) (s2, s10) (s3, s4) (s3, s5) (s3, s6) (s3, s7) (s3, s8) (
s3, s9) (s3, s10) (s4, s5) (s4, s6) (s4, s7) (s4, s8) (s4, s9) (s4, s10)
(s5, s6) (s5, s7) (s5, s8) (s5, s9) (s5, s10) (s6, s7) (s6, s8) (s6, s9)
(s6, s10) (s7, s8) (s7, s9) (s7, s10) (s8, s9) (s8, s10) (s9, s10)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Starting CLI:
mininet> 
```

Figure 13. Start Mininet topology with 10 switches

After the initiation of each Mininet topology, the controller auto-calculates the minimum spanning tree (executes the Mininet CLI).

Urls API Overview

We have implemented some scripts which help us to understand the results and the performance of java code. We have the following scripts and curl commands:

- The url http://127.0.0.1:8080/wm/mst/mstedges/json gives us the computed edges.
- The url http://127.0.0.1:8080/wm/mst/redundantedges/json gives us the computed redundant edges which we will not take them account when we make the traffic engineering.
- The url http://127.0.0.1:8080/wm/mst/topocosts/json gives us the computed topology cost for all the topology and shows us the cost of every LSP.
- The url http://127.0.0.1:8080/wm/mst/topoedges/json gives us all the edges of topology and each their cost.

To conclude the script viewMSTapis.sh allows us to know every computed cost of topology via floodlight REST API. The script has 4 options. When we call the case of mstedges, the script returns us all the cost of every edge in the network. When we call the topocosts option, the script returns us the cost of all topology. Then we have the option of topoedges which shows us the cost of edges which we will use to make the TE. Final the option of redundantedges gives us the edges which will not be use to TE.

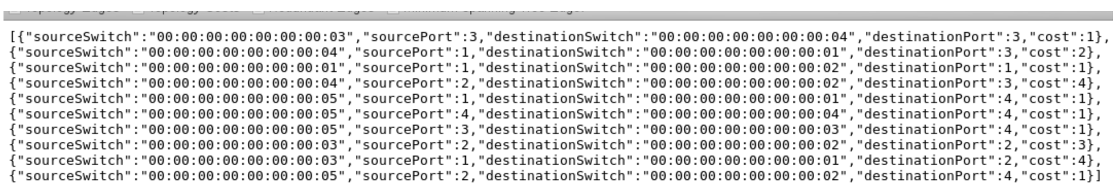
6.1 Show topology and edge costs

The topology can be shown with the following command:

```
$ ./viewMSTapis.sh -a topoedges
```

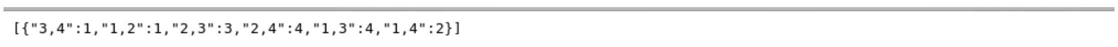
The costs for all edges of the topology are as shown by command:

```
$ ./viewMSTapis.sh -a topocosts
```



```
[{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":3,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":2},
{"sourceSwitch":"00:00:00:00:00:00:00:01","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":1,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":4},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":3},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":2,"cost":4},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":1}]
```

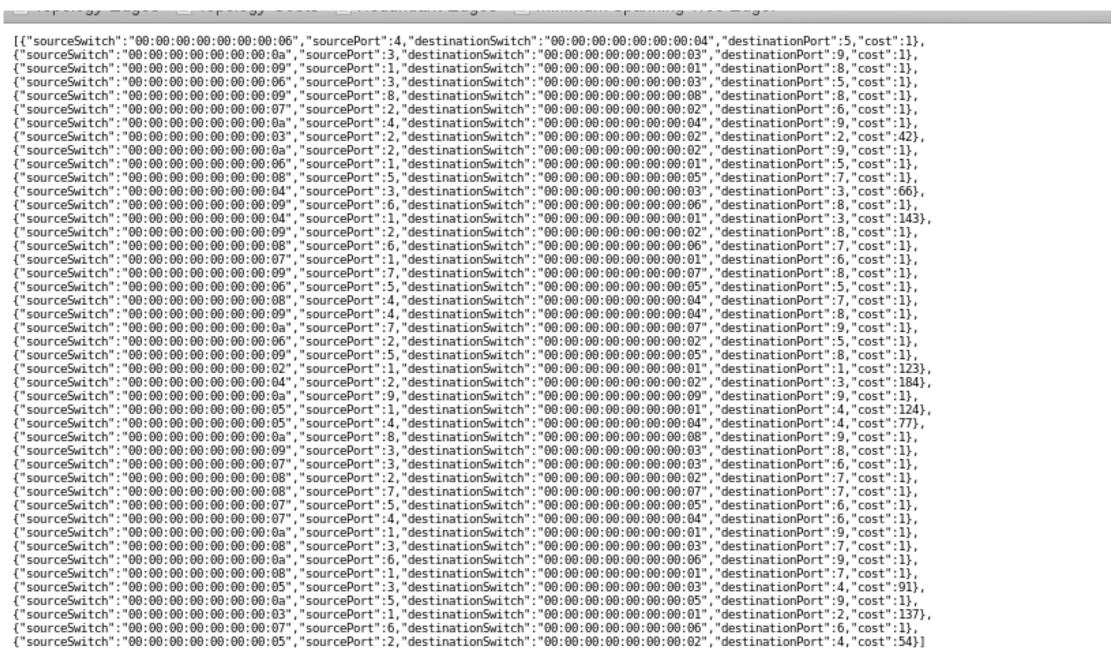
Figure 14. Show Mininet topology with 5 switches



```
[{"3,4":1,"1,2":1,"2,3":3,"2,4":4,"1,3":4,"1,4":2}]
```

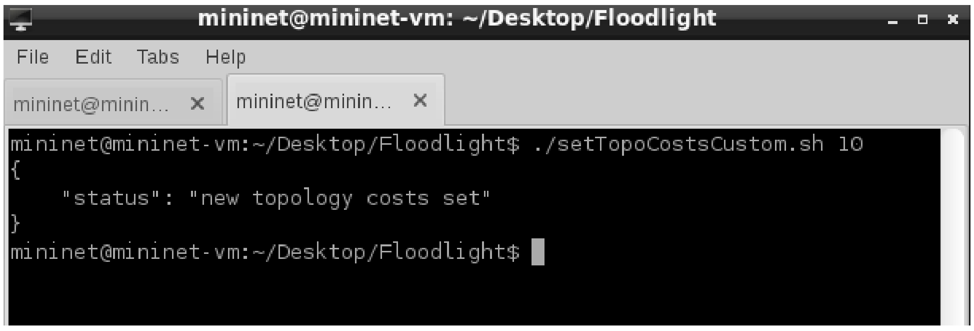
Figure 15. Show Mininet link costs with 5 switches

For the 10-switch topology link costs had to be set by the user, since the initial values for the edges were not realistic.



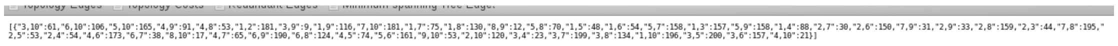
```
[{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":5,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":5,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":6,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":42},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":5,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":3,"cost":66},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":143},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":6,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":5,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":5,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:02","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":1,"cost":123},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":184},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":9,"destinationSwitch":"00:00:00:00:00:00:00:09","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":124},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":77},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":8,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":6,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":6,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":6,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":7,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":91},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":9,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":2,"cost":137},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":6,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":54}]
```

Figure 16. Show Mininet topology with 10 switches – not good initial values.



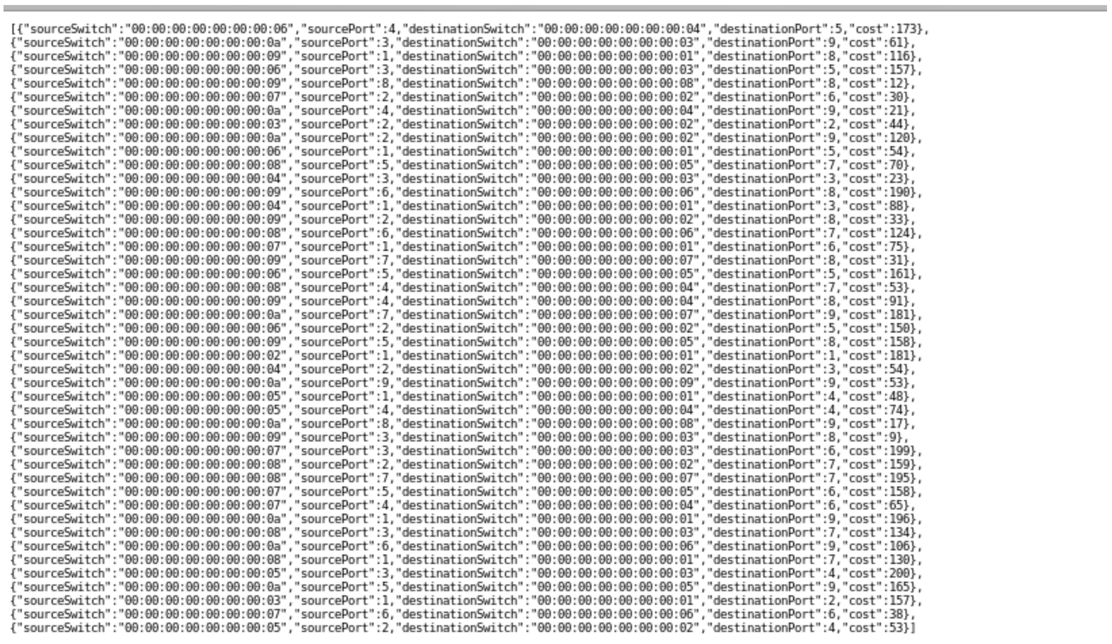
```
mininet@mininet-vm:~/Desktop/Floodlight$ ./setTopoCostsCustom.sh 10
{
    "status": "new topology costs set"
}
mininet@mininet-vm:~/Desktop/Floodlight$ 
```

Figure 17. Set Mininet topology costs for 10 switch network.



```
[{"3,10":61,"6,10":106,"5,10":165,"4,9":91,"4,8":53,"1,2":181,"3,9":9,"1,9":116,"7,10":181,"1,7":75,"1,8":130,"8,9":12,"5,8":70,"1,5":48,"1,6":54,"5,7":158,"1,3":157,"5,9":158,"1,4":88,"2,7":30,"2,6":150,"7,9":31,"2,9":33,"2,8":159,"2,3":44,"7,8":195,"
2,5":53,"2,4":54,"4,6":173,"6,7":38,"8,10":17,"4,7":65,"6,9":190,"6,8":124,"4,5":74,"5,6":161,"9,10":53,"2,10":120,"3,4":23,"3,7":199,"3,8":134,"1,10":196,"3,5":200,"3,6":157,"4,10":21}]
```

Figure 18. Show new Mininet link costs with 10 switches



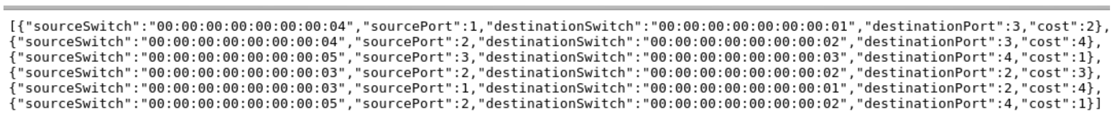
```
[{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":5,"cost":173},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":9,"cost":61},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":8,"cost":116},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":5,"cost":157},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":8,"cost":12},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":6,"cost":30},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":9,"cost":21},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":44},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":9,"cost":120},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":5,"cost":54},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":7,"cost":70},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":3,"cost":23},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":8,"cost":190},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":88},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":8,"cost":33},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":7,"cost":124},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":6,"cost":75},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":8,"cost":31},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":5,"cost":161},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":7,"cost":53},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":8,"cost":91},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":9,"cost":181},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":5,"cost":150},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":8,"cost":158},
{"sourceSwitch":"00:00:00:00:00:00:00:02","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":1,"cost":181},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":54},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":9,"destinationSwitch":"00:00:00:00:00:00:00:09","destinationPort":9,"cost":53},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":48},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":74},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":9,"cost":17},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":8,"cost":9},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":6,"cost":199},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":7,"cost":159},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":7,"cost":195},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":6,"cost":158},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":6,"cost":65},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":9,"cost":196},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":7,"cost":134},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":9,"cost":106},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":7,"cost":130},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":200},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":9,"cost":165},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":2,"cost":157},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":6,"cost":38},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":53}]
```

Figure 19. Show new Mininet topology with 10 switches

After Kruskal algorithm implementation, there are some redundant edges for both topologies.

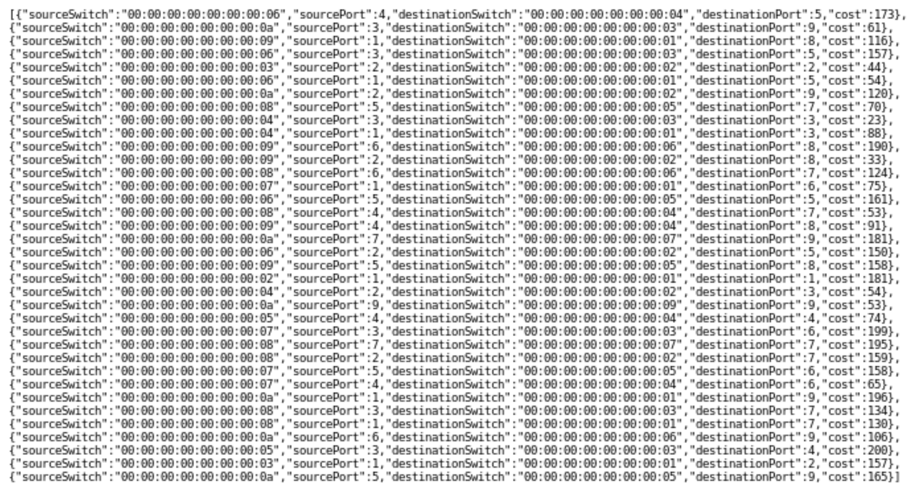
Redundant edges can be shown with the following command:

```
$ ./viewMSTapis.sh -a redundantedges
```



```
[{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":2},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":4},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":3},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":2,"cost":4},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":1}]
```

Figure 20. Show redundant edges 5-switch topology



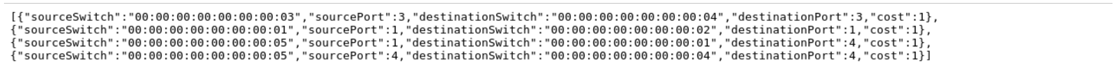
```
[{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":5,"cost":173},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":9,"cost":61},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":8,"cost":116},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":5,"cost":157},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":44},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":5,"cost":54},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":9,"cost":120},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":7,"cost":70},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":3,"cost":23},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":88},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":8,"cost":190},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":8,"cost":33},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":7,"cost":124},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":6,"cost":75},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":5,"cost":161},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":7,"cost":53},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":8,"cost":91},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":9,"cost":181},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":5,"cost":150},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":8,"cost":158},
{"sourceSwitch":"00:00:00:00:00:00:00:02","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":1,"cost":181},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":54},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":9,"destinationSwitch":"00:00:00:00:00:00:00:09","destinationPort":9,"cost":53},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":74},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":6,"cost":199},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":7,"cost":195},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":7,"cost":159},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":6,"cost":158},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":6,"cost":65},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":9,"cost":196},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":7,"cost":134},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":7,"cost":130},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":9,"cost":106},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":200},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":2,"cost":157},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":9,"cost":165}]
```

Figure 21. Show redundant edges 10-switch topology

6.2 Minimum spanning trees

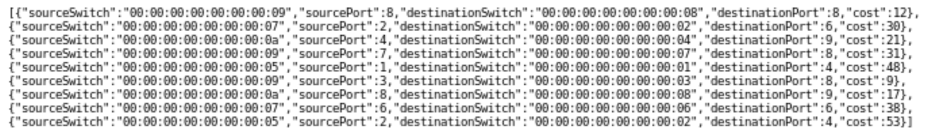
MST (Minimum Spanning Tree) edges can be shown with the following command:

```
$ ./viewMSTapis.sh -a mstedges
```



```
[{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":3,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:01","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":1,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":1}]
```

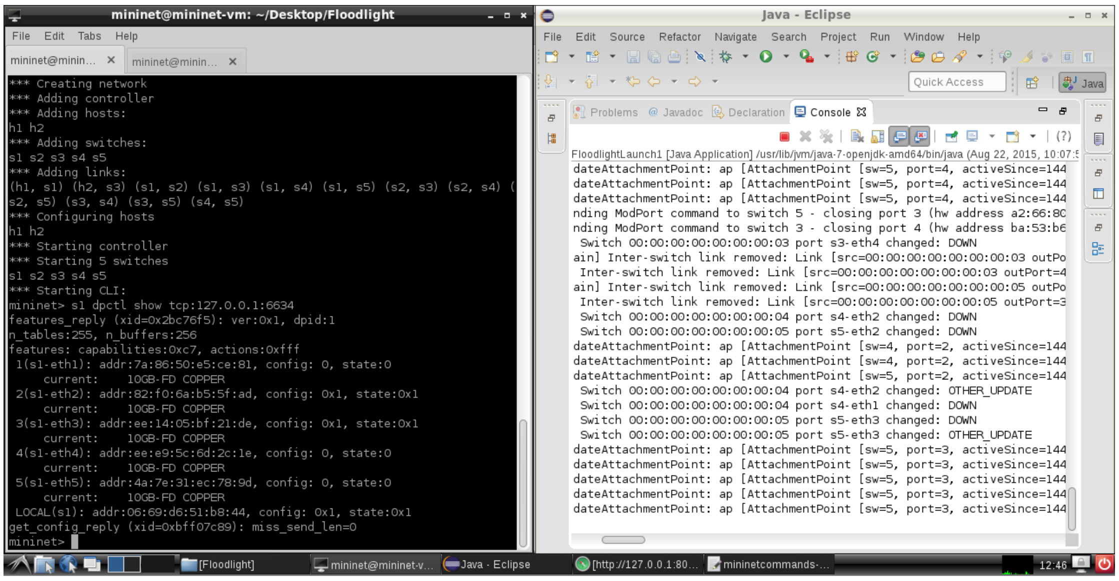
Figure 22. Show MST for 5-switch topology



```
[{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":8,"cost":12},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":6,"cost":30},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":9,"cost":21},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":8,"cost":31},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":48},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":8,"cost":9},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":9,"cost":17},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":6,"cost":38},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":53}]
```

Figure 23. Show MST for 10-switch topology

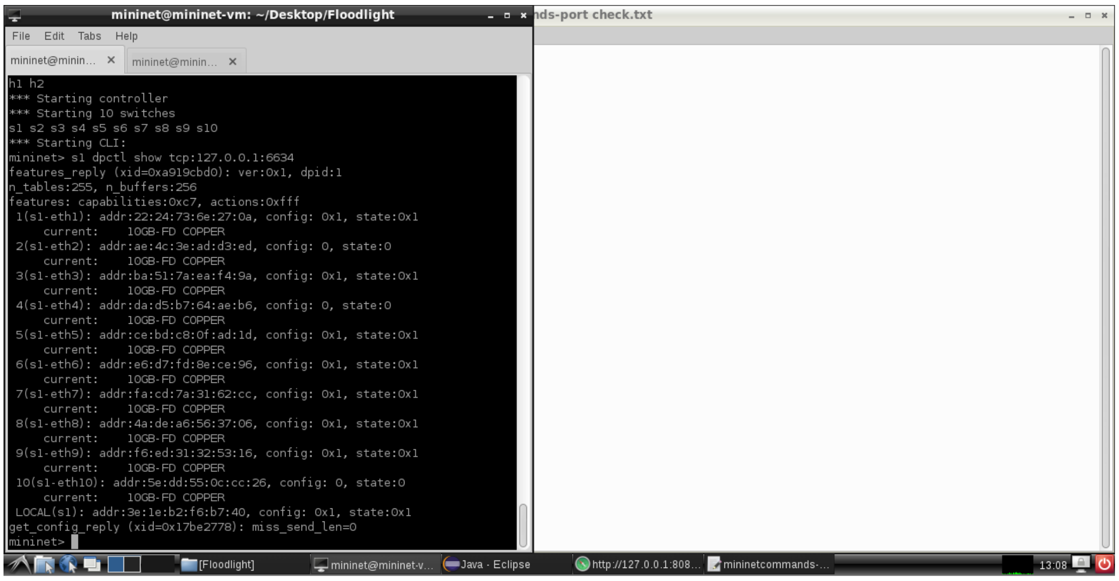
As shown in the following figure, we can see the features of each switch by calling it in Mininet.



```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s3) (s1, s2) (s1, s3) (s1, s4) (s1, s5) (s2, s3) (s2, s4) (
s2, s5) (s3, s4) (s3, s5) (s4, s5)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 5 switches
s1 s2 s3 s4 s5
*** Starting CLI:
mininet> s1 dpctl show tcp:127.0.0.1:6634
features_reply (xid=0x2bc76f5): ver:0x1, dpid:1
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
 1(s1-eth1): addr:7a:86:50:e5:ce:81, config: 0, state:0
     current:    10GB-FD COPPER
 2(s1-eth2): addr:82:f0:6a:b5:5f:ad, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 3(s1-eth3): addr:ee:14:05:bf:21:de, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 4(s1-eth4): addr:ee:e9:5c:6d:2c:1e, config: 0, state:0
     current:    10GB-FD COPPER
 5(s1-eth5): addr:4a:7e:31:ec:78:9d, config: 0, state:0
     current:    10GB-FD COPPER
 LOCAL(s1): addr:06:69:d6:51:b8:44, config: 0x1, state:0x1
get_config_reply (xid=0xbff07c89): miss_send_len=0
mininet>
```

```
FloodlightLaunch1 [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Aug 22, 2015, 10:07:5
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=4, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=4, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=4, activeSince=144
nding ModPort command to switch 5 - closing port 3 (hw address a2:66:8C
nding ModPort command to switch 3 - closing port 4 (hw address ba:53:b6
 Switch 00:00:00:00:00:00:00:03 port s3-eth4 changed: DOWN
ain] Inter-switch link removed: Link [src=00:00:00:00:00:00:00:03 outPo
 Inter-switch link removed: Link [src=00:00:00:00:00:00:00:03 outPort=4
ain] Inter-switch link removed: Link [src=00:00:00:00:00:00:00:05 outPo
 Inter-switch link removed: Link [src=00:00:00:00:00:00:00:05 outPort=3
 Switch 00:00:00:00:00:00:00:04 port s4-eth2 changed: DOWN
 Switch 00:00:00:00:00:00:00:05 port s5-eth2 changed: DOWN
dateAttachmentPoint: ap [AttachmentPoint [sw=4, port=2, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=4, port=2, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=2, activeSince=144
 Switch 00:00:00:00:00:00:00:04 port s4-eth2 changed: OTHER_UPDATE
 Switch 00:00:00:00:00:00:00:04 port s4-eth1 changed: DOWN
 Switch 00:00:00:00:00:00:00:05 port s5-eth3 changed: DOWN
 Switch 00:00:00:00:00:00:00:05 port s5-eth3 changed: OTHER_UPDATE
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=3, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=3, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=3, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=3, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=3, activeSince=144
```

Figure 24. Show s1 features in 5-switch topology

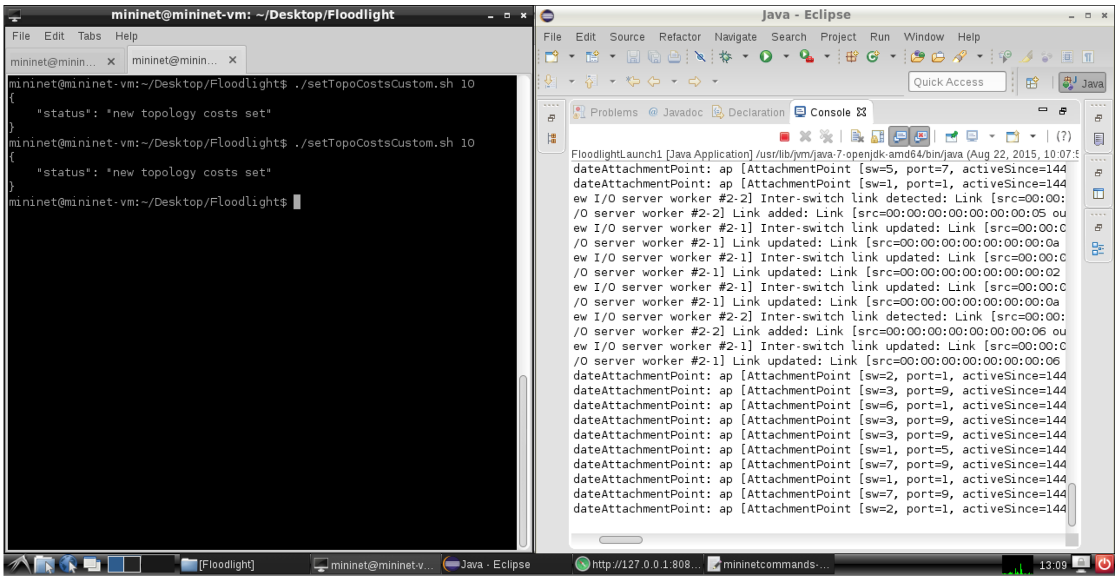


```
h1 h2
*** Starting controller
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Starting CLI:
mininet> s1 dpctl show tcp:127.0.0.1:6634
features_reply (xid=0xa919cbd0): ver:0x1, dpid:1
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
 1(s1-eth1): addr:22:24:73:6e:27:0a, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 2(s1-eth2): addr:ae:4c:3e:ad:d3:ed, config: 0, state:0
     current:    10GB-FD COPPER
 3(s1-eth3): addr:ba:51:7a:ea:f4:9a, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 4(s1-eth4): addr:da:d5:b7:64:ae:b6, config: 0, state:0
     current:    10GB-FD COPPER
 5(s1-eth5): addr:ce:bd:c8:0f:ad:1d, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 6(s1-eth6): addr:e6:d7:fd:8e:ce:96, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 7(s1-eth7): addr:fa:cd:7a:31:62:cc, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 8(s1-eth8): addr:4a:de:a6:56:37:06, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 9(s1-eth9): addr:f6:ed:31:32:53:16, config: 0x1, state:0x1
     current:    10GB-FD COPPER
 10(s1-eth10): addr:5e:dd:55:0c:cc:26, config: 0, state:0
     current:    10GB-FD COPPER
 LOCAL(s1): addr:3e:1e:b2:f6:b7:40, config: 0x1, state:0x1
get_config_reply (xid=0x17be2778): miss_send_len=0
mininet>
```

Figure 25. Show s1 features in 10-switch topology

6.3 Setting new costs

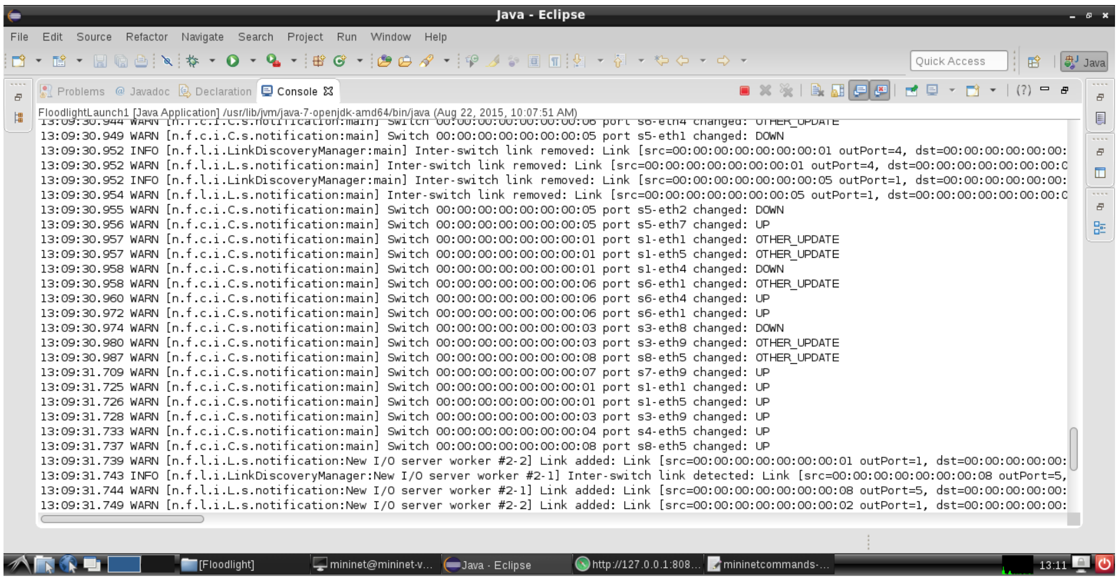
For the 10-switch topology we set new costs without quiting the network and we can see that Floodlight controller updates the links in real time.



```
mininet@mininet-vm:~/Desktop/Floodlight$ ./setTopoCostsCustom.sh 10
{
    "status": "new topology costs set"
}
mininet@mininet-vm:~/Desktop/Floodlight$ ./setTopoCostsCustom.sh 10
{
    "status": "new topology costs set"
}
mininet@mininet-vm:~/Desktop/Floodlight$
```

```
FloodlightLaunch1 [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Aug 22, 2015, 10:07:5
dateAttachmentPoint: ap [AttachmentPoint [sw=5, port=7, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=1, port=1, activeSince=144
ew I/O server worker #2-2] Inter-switch link detected: Link [src=00:00:
/O server worker #2-2] Link added: Link [src=00:00:00:00:00:00:00:05 ou
ew I/O server worker #2-1] Inter-switch link updated: Link [src=00:00:0
/O server worker #2-1] Link updated: Link [src=00:00:00:00:00:00:00:0a
ew I/O server worker #2-1] Inter-switch link updated: Link [src=00:00:0
/O server worker #2-1] Link updated: Link [src=00:00:00:00:00:00:00:02
ew I/O server worker #2-1] Inter-switch link updated: Link [src=00:00:0
/O server worker #2-1] Link updated: Link [src=00:00:00:00:00:00:00:0a
ew I/O server worker #2-2] Inter-switch link detected: Link [src=00:00:
/O server worker #2-2] Link added: Link [src=00:00:00:00:00:00:00:06 ou
ew I/O server worker #2-1] Inter-switch link updated: Link [src=00:00:0
/O server worker #2-1] Link updated: Link [src=00:00:00:00:00:00:00:06
dateAttachmentPoint: ap [AttachmentPoint [sw=2, port=1, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=3, port=9, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=6, port=1, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=3, port=9, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=3, port=9, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=1, port=5, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=7, port=9, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=1, port=1, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=7, port=9, activeSince=144
dateAttachmentPoint: ap [AttachmentPoint [sw=2, port=1, activeSince=144
```

Figure 26. Set new costs in 10-switch topology

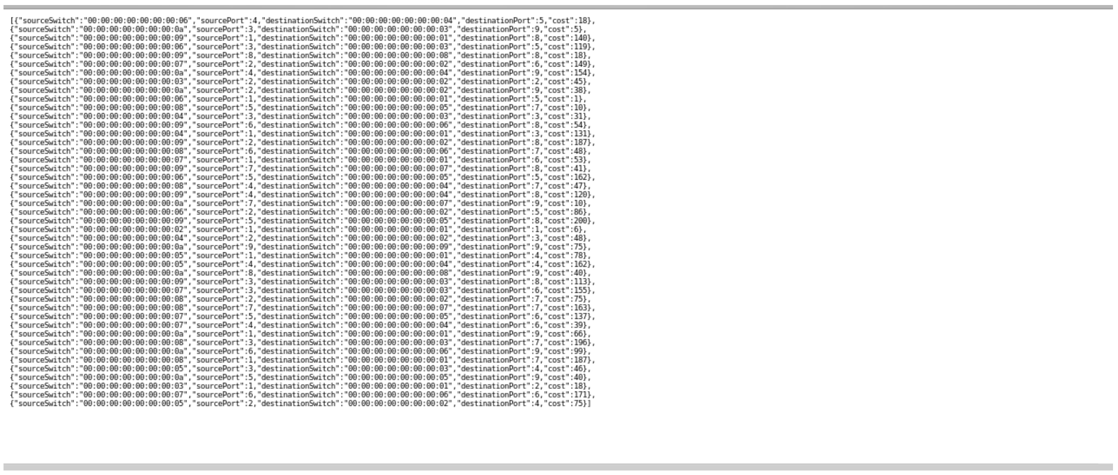


```
FloodlightLaunch1 [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Aug 22, 2015, 10:07:51 AM)
13:09:30.949 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:05 port s5-eth1 changed: DOWN
13:09:30.952 INFO [n.f.l.i.LinkDiscoveryManager:main] Inter-switch link removed: Link [src=00:00:00:00:00:00:00:01 outPort=4, dst=00:00:00:00:00:00:
13:09:30.952 WARN [n.f.l.i.L.s.notification:main] Inter-switch link removed: Link [src=00:00:00:00:00:00:00:01 outPort=4, dst=00:00:00:00:00:00:0
13:09:30.952 INFO [n.f.l.i.LinkDiscoveryManager:main] Inter-switch link removed: Link [src=00:00:00:00:00:00:00:05 outPort=1, dst=00:00:00:00:00:00:
13:09:30.954 WARN [n.f.l.i.L.s.notification:main] Inter-switch link removed: Link [src=00:00:00:00:00:00:00:05 outPort=1, dst=00:00:00:00:00:00:0
13:09:30.955 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:05 port s5-eth2 changed: DOWN
13:09:30.956 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:05 port s5-eth7 changed: UP
13:09:30.957 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 port s1-eth1 changed: OTHER_UPDATE
13:09:30.957 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 port s1-eth5 changed: OTHER_UPDATE
13:09:30.958 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 port s1-eth4 changed: DOWN
13:09:30.958 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:06 port s6-eth1 changed: OTHER_UPDATE
13:09:30.960 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:06 port s6-eth4 changed: UP
13:09:30.972 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:06 port s6-eth1 changed: UP
13:09:30.974 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:03 port s3-eth8 changed: DOWN
13:09:30.980 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:03 port s3-eth9 changed: OTHER_UPDATE
13:09:30.987 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:08 port s8-eth5 changed: OTHER_UPDATE
13:09:31.709 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:07 port s7-eth9 changed: UP
13:09:31.725 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 port s1-eth1 changed: UP
13:09:31.726 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 port s1-eth5 changed: UP
13:09:31.728 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:03 port s3-eth9 changed: UP
13:09:31.733 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:04 port s4-eth5 changed: UP
13:09:31.737 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:08 port s8-eth5 changed: UP
13:09:31.739 WARN [n.f.l.i.L.s.notification:New I/O server worker #2-2] Link added: Link [src=00:00:00:00:00:00:00:01 outPort=1, dst=00:00:00:00:00:
13:09:31.743 INFO [n.f.l.i.LinkDiscoveryManager:New I/O server worker #2-1] Inter-switch link detected: Link [src=00:00:00:00:00:00:00:08 outPort=5,
13:09:31.744 WARN [n.f.l.i.L.s.notification:New I/O server worker #2-1] Link added: Link [src=00:00:00:00:00:00:00:08 outPort=5, dst=00:00:00:00:00:
13:09:31.749 WARN [n.f.l.i.L.s.notification:New I/O server worker #2-2] Link added: Link [src=00:00:00:00:00:00:00:02 outPort=1, dst=00:00:00:00:00:
```

Figure 27. Controller updates links in real time for 10-switch topology

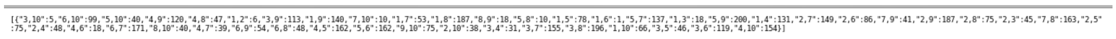
The resulting new edges and costs for the 10-switch topology and costs are shown in the following.

We also present the redundant edges.



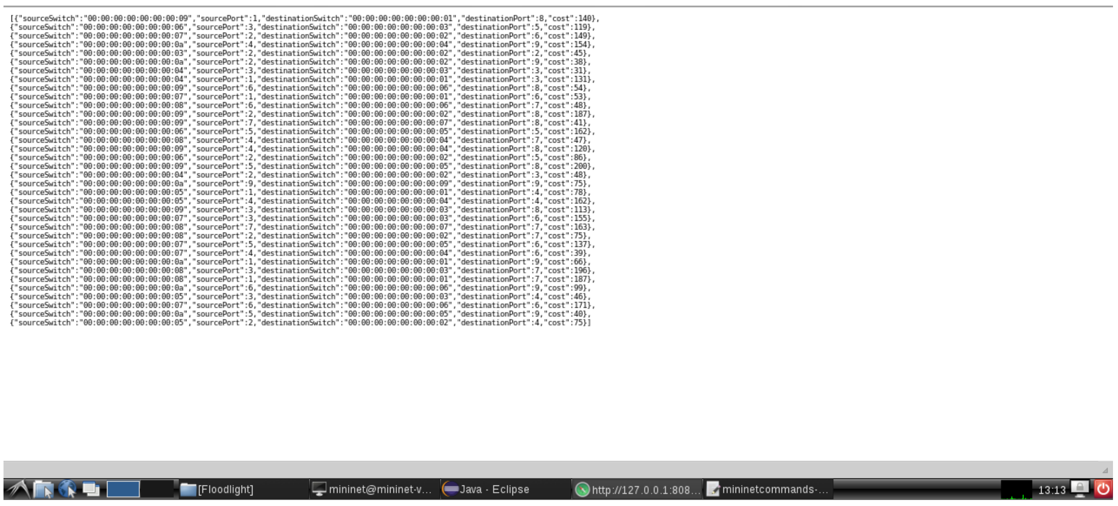
```
[{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":5,"cost":18},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":9,"cost":5},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":8,"cost":140},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":5,"cost":119},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":8,"cost":18},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":6,"cost":149},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":9,"cost":154},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":45},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":9,"cost":38},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":5,"cost":1},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":7,"cost":10},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":3,"cost":31},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":8,"cost":54},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":131},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":8,"cost":187},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":7,"cost":48},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":6,"cost":53},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":8,"cost":41},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":5,"cost":162},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":7,"cost":47},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":8,"cost":120},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":9,"cost":10},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":5,"cost":86},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":8,"cost":200},
{"sourceSwitch":"00:00:00:00:00:00:00:02","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":1,"cost":6},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":48},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":9,"destinationSwitch":"00:00:00:00:00:00:00:09","destinationPort":9,"cost":75},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":78},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":162},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":8,"destinationSwitch":"00:00:00:00:00:00:00:08","destinationPort":9,"cost":40},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":8,"cost":113},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":6,"cost":155},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":7,"cost":75},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":7,"cost":163},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":6,"cost":137},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":6,"cost":39},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":9,"cost":66},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":7,"cost":196},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":9,"cost":99},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":7,"cost":187},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":46},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":9,"cost":40},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":2,"cost":18},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":6,"cost":171},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":75}]
```

Figure 28. Show Mininet topology with 10 switches - 2



```
[{"3,10":5,"6,10":99,"5,10":40,"4,9":120,"4,8":47,"1,2":6,"3,9":113,"1,9":140,"7,10":10,"1,7":53,"1,8":187,"8,9":18,"5,8":10,"1,5":78,"1,6":1,"5,7":137,"1,3":18,"5,9":200,"1,4":131,"2,7":149,"2,6":86,"7,9":41,"2,9":187,"2,8":75,"2,3":45,"7,8":163,"2,5"
:75,"2,4":48,"4,6":18,"6,7":171,"8,10":40,"4,7":39,"6,9":54,"6,8":48,"4,5":162,"5,6":162,"9,10":75,"2,10":38,"3,4":31,"3,7":155,"3,8":196,"1,10":66,"3,5":46,"3,6":119,"4,10":154}]
```

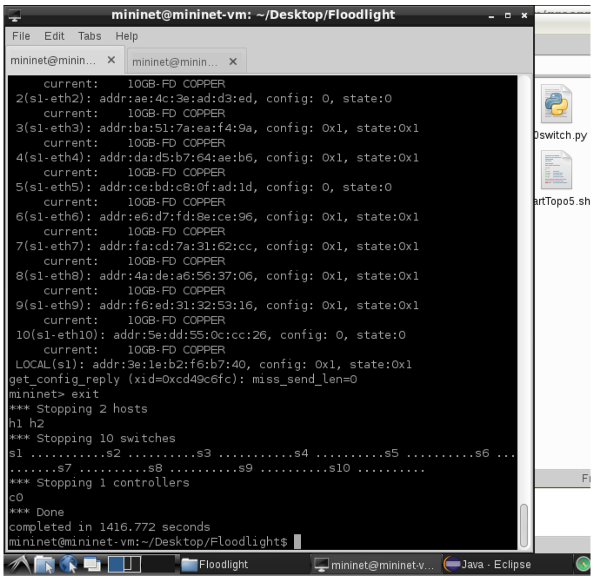
Figure 29. Show Mininet edge costs with 10 switches – 2



```
[{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":8,"cost":140},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":5,"cost":119},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":6,"cost":149},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":9,"cost":154},
{"sourceSwitch":"00:00:00:00:00:00:00:03","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":2,"cost":45},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":9,"cost":38},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":3,"cost":31},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":3,"cost":131},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":8,"cost":54},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":6,"cost":53},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":7,"cost":48},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":8,"cost":187},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":8,"cost":41},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":5,"cost":162},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":7,"cost":47},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":8,"cost":120},
{"sourceSwitch":"00:00:00:00:00:00:00:06","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":5,"cost":86},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":8,"cost":200},
{"sourceSwitch":"00:00:00:00:00:00:00:04","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":3,"cost":48},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":9,"destinationSwitch":"00:00:00:00:00:00:00:09","destinationPort":9,"cost":75},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":4,"cost":78},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":4,"cost":162},
{"sourceSwitch":"00:00:00:00:00:00:00:09","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":8,"cost":113},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":6,"cost":155},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":7,"destinationSwitch":"00:00:00:00:00:00:00:07","destinationPort":7,"cost":163},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":7,"cost":75},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":6,"cost":137},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":4,"destinationSwitch":"00:00:00:00:00:00:00:04","destinationPort":6,"cost":39},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":9,"cost":66},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":7,"cost":196},
{"sourceSwitch":"00:00:00:00:00:00:00:08","sourcePort":1,"destinationSwitch":"00:00:00:00:00:00:00:01","destinationPort":7,"cost":187},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":9,"cost":99},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":3,"destinationSwitch":"00:00:00:00:00:00:00:03","destinationPort":4,"cost":46},
{"sourceSwitch":"00:00:00:00:00:00:00:07","sourcePort":6,"destinationSwitch":"00:00:00:00:00:00:00:06","destinationPort":6,"cost":171},
{"sourceSwitch":"00:00:00:00:00:00:00:0a","sourcePort":5,"destinationSwitch":"00:00:00:00:00:00:00:05","destinationPort":9,"cost":40},
{"sourceSwitch":"00:00:00:00:00:00:00:05","sourcePort":2,"destinationSwitch":"00:00:00:00:00:00:00:02","destinationPort":4,"cost":75}]
```

Figure 30. Show Mininet redundant edges with 10 switches – 2

After execution the topology is terminated in Mininet with exit command.



```
    current:    10GB-FD COPPER
 2(s1-eth2): addr:ae:4c:3e:ad:d3:ed, config: 0, state:0
    current:    10GB-FD COPPER
 3(s1-eth3): addr:ba:51:7a:ea:f4:9a, config: 0x1, state:0x1
    current:    10GB-FD COPPER
 4(s1-eth4): addr:da:d5:b7:64:ae:b6, config: 0x1, state:0x1
    current:    10GB-FD COPPER
 5(s1-eth5): addr:ce:bd:c8:0f:ad:1d, config: 0, state:0
    current:    10GB-FD COPPER
 6(s1-eth6): addr:e6:d7:fd:8e:ce:96, config: 0x1, state:0x1
    current:    10GB-FD COPPER
 7(s1-eth7): addr:fa:cd:7a:31:62:cc, config: 0x1, state:0x1
    current:    10GB-FD COPPER
 8(s1-eth8): addr:4a:de:a6:56:37:06, config: 0x1, state:0x1
    current:    10GB-FD COPPER
 9(s1-eth9): addr:f6:ed:31:32:53:16, config: 0x1, state:0x1
    current:    10GB-FD COPPER
 10(s1-eth10): addr:5e:dd:55:0c:cc:26, config: 0, state:0
    current:    10GB-FD COPPER
 LOCAL(s1): addr:3e:1e:b2:f6:b7:40, config: 0x1, state:0x1
get_config_reply (xid=0xcd49c6fc): miss_send_len=0
mininet> exit
*** Stopping 2 hosts
h1 h2
*** Stopping 10 switches
s1 ..........s2 ..........s3 ..........s4 ..........s5 ..........s6 ...
.......s7 ..........s8 ..........s9 ..........s10 ..........
*** Stopping 1 controllers
c0
*** Done
completed in 1416.772 seconds
mininet@mininet-vm:~/Desktop/Floodlight$ 
```

Figure 31. Mininet topology termination.

6.4 Running time

Kruskal algorithm has complexity O (|E| log |V|), where E is the number of edges and V the number of graph peaks or switches. Kruskal's algorithm performs:

- a heap building operation on an array of E entries, which takes O(E) time;
- at most E minimum exctraction operations on a heap with at most E entries, each taking O(log E) = O(log V^2) = O(log V) time, for a total of O(E log V) time
- V set making operations, each taking O(1) time, for a total of O(n) time
- at most 2E Find operations (two per edge), and at most V − 1 union operations, for a total of O(E log V) time.

So, the total running time of the algorithm is O(E)+O(E log V)+O(V)+O(E logV) = O(E log V).

The 5-switch topology contains 2 hosts and 5 switches. With this configuration, 12 links/edges are created. Hence execution time can be estimated as 12 * log7 = 10.14.

The 10-switch topology contains 2 hosts and 10 switches. With this configuration, 47 links/edges are created. Hence execution time can be estimated as 47 * log17 = 57.83.

6.5 Ports manage via rest API for energy efficient

The implementation of the code gives us the opportunity via traffic engineering to open and close ports in order to have the minimum energy cost on the network.

This is the major point of the code and we are going to analyze below the scripts and how they works.

When we execute the algorithm we get the MST that we can see below via mininet.

```
mininet> s4 dpctl show tcp:127.0.0.1:6637
features_reply (xid=0x4a7cacc5): ver:0x1, dpid:4
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
1(s4-eth1): addr:a2:8f:c2:a8:6d:57, config: 0, state:0
    current:    10GB-FD COPPER
2(s4-eth2): addr:ba:d8:da:a1:26:5b, config: 0x1, state:0x1
    current:    10GB-FD COPPER
3(s4-eth3): addr:0e:1d:34:0a:18:b2, config: 0, state:0
    current:    10GB-FD COPPER
LOCAL(s4): addr:82:45:0b:bc:e8:44, config: 0x1, state:0x1
get_config_reply (xid=0xe67cf3b9): miss_send_len=0
```

From the above example it is possible to verify that the port on switch 4 has been closed.

In the project it has been implemented a script which change the cost of the LSPs

```
./setTopoCosts.sh
{
  "status": "new topology costs set"
}

$ ./viewMSTapis.sh -a topocosts
[
  {
    "1,2": 10,
    "1,3": 40,
    "1,4": 20,
    "2,3": 30,
    "2,4": 10,
    "3,4": 40
  }
]
```

The code has recomputed the MST, and we can check this with the script:

```
$ ./viewMSTapis.sh -a mstedges
[
  {
    "cost": 10,
    "destinationPort": 1,
    "destinationSwitch": "00:00:00:00:00:00:00:02",
    "sourcePort": 1,
    "sourceSwitch": "00:00:00:00:00:00:00:01"
  },
  {
    "cost": 10,
    "destinationPort": 3,
    "destinationSwitch": "00:00:00:00:00:00:00:02",
    "sourcePort": 2,
    "sourceSwitch": "00:00:00:00:00:00:00:04"
  },
  {
    "cost": 30,
    "destinationPort": 2,
    "destinationSwitch": "00:00:00:00:00:00:00:03",
    "sourcePort": 2,
    "sourceSwitch": "00:00:00:00:00:00:00:02"
  }
]
```

The new MST has been deployed, as we can see below:

```
mininet> s4 dpctl show tcp:127.0.0.1:6637
features_reply (xid=0x461fa8c): ver:0x1, dpid:4
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
       1(s4-eth1): addr:a2:8f:c2:a8:6d:57, config: 0x1, state:0x1
current:    10GB-FD COPPER
2(s4-eth2): addr:ba:d8:da:a1:26:5b, config: 0, state:0
    current:    10GB-FD COPPER
3(s4-eth3): addr:0e:1d:34:0a:18:b2, config: 0x1, state:0x1
    current:    10GB-FD COPPER
LOCAL(s4): addr:82:45:0b:bc:e8:44, config: 0x1, state:0x1
get_config_reply (xid=0x59b0cd47): miss_send_len=0
```

As we can see above we have the port 2 which is active while port 1 and 3 are down.

7. Conclusions

Energy saving nowadays is the most important concern in all societies. As understood, the concerns and the effort to mitigate excess energy consumption from Internet and its applications is an important fact, since its use has become an important part in everyday life.

Software-defined network architecture is still a new method to create and manage computer networks, which are still being developed and tested. Over time protocols that support it (like OpenFlow) become more mature and reliable, and more controllers are developed in different programming languages, in addition to NOX, such as Beacon or Floodlight, offering more and more features to create centralized, functional and reliable computer networks.

Hence, in seeking to address the phenomenon of resource optimization in the context of this thesis, a mechanism was implemented by Floodlight controller, which communicates interactively with the network devices and can continually exchange messages. The controller regulates the topology of the network and collects statistics on load through routers. Subsequently data are collected by an optimization software that supports optimization through Kruskal algorithm and Minimum Spanning Tree, to find the most economical routing.

8. References

1. ONF White Paper, "Software-Defined Networking: The New Norm for Networks", April 2012

2. Open Networking Foundation (ONF), "SDN Architecture Overview", Version 1.0, December 2013

3. Fei Hu, "Network Innovation through Openflow and SDN Principles and Design", CRC Press, 2014

4. OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05), October 2013

5. Component- Based Software Defined Networking Framework: Build SDN Agilely, http://osrg.github.io/ryu/

6. Introduction to Mininet https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

7. Ruud Louwersheimer, "Implementing Anycast in IPv4 Networks", INS, June 2004

8. Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzellery, Martin Casado, Nick McKeowny, Guru Parulkary *Flowvisor: A network virtualization layer* 2009

9. Saurav Das, Yiannis Yiakoumis, Guru Parulkar, Nick McKeown, Preeti Singh, Daniel Getachew, Premal Dinesh Desai *Application-Aware Aggregation and Traffic Engineering in a Converged Packet-Circuit Network*

10. Hilmi E. Egilmez, Burak Gorkemli, A. Murat Tekalp, Seyhan Civanlar *SCALABLE VIDEO STREAMING OVER OPENFLOW NETWORKS: AN OPTIMIZATION FRAMEWORK FOR QOS ROUTING*

11. Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, Rob Sherwood *On Controller Perormance in Software Defined Networks*

URLS

[1] http://www.projectfloodlight.org/

[2] https://github.com

[3] http://en.wikipedia.org/

[4] http://www.networkcomputing.com/

[5] https://groups.google.com/

[6] http://searchsdn.techtarget.com/

[7] http://archive.openflow.org/

[8] http://networkstatic.net/

[9] http://thenewstack.io/

[10]http://ieeexplore.ieee.org/

[11]http://www.cisco.com/

[12]https://www.sdxcentral.com

[13]http://sdntutorials.com/

[14]http://www.webopedia.com

[15]http://sdn101.com/

[16]https://n40lab.wordpress.com/

[17]https://www.passeidireto.com

[18]http://algs4.cs.princeton.edu/

[19]http://cs.stackexchange.com/

[20]http://stackoverflow.com/

[21]http://www.channelworld.in/

[22]https://www.citrix.com/

[23]http://www.sciencedirect.com/

[24]https://www.opennetworking.org/

[25]https://www.coursera.org/

[26]http://ieeexplore.ieee.org/

[27]http://www.networkworld.com/

[28]http://archive.openflow.org/wp/learnmore/

[29]http://www.stoimen.com

[30]http://scanftree.com

[31]http://bigswitch.com/

[32]http://www.sciencedirect.com/

[33]http://flowgrammable.org/