



ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ  
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ, ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ  
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Πτυχιακή Εργασία

Σχεδιασμός, υλοποίηση και αξιολόγηση δικτύων, καθοριζόμενων από λογισμικό

Ντούβλης Στέφανος

Επιβλέπων: Καλόξυλος Αλέξανδρος, Αναπληρωτής Καθηγητής

Τρίπολη  
Οκτώβριος 2021



HELLENIC DEMOCRACY UNIVERSITY OF PELOPONNESE  
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS, FACULTY OF  
ECONOMICS AND TECHNOLOGY

## THESIS

Design, implementation & evaluation of software defined networks

Ntouvlis Stefanos

Supervisor: Kaloxylos Alexandros, Associate Professor

Tripoli  
October 2021

## Πτυχιακή

Σχεδιασμός, υλοποίηση και αξιολόγηση δικτύων, καθοριζόμενων από λογισμικό

Ντούβλης Στέφανος  
Α.Μ 2022201700134

ΕΠΙΒΛΕΠΩΝ: Καλόξυλος Αλέξανδρος, Αναπληρωτής Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:  
Καλόξυλος Αλέξανδρος, Αναπληρωτής Καθηγητής  
Γιώργιος Τσούλος, Καθηγητής

1 Νοεμβρίου 2021

# THESIS

Design, implementation & evaluation of software defined networks

Ntouvlis Stefanos  
S.N 2022201700134

SUPERVISOR: Kaloxylos Alexandros, Associate Professor

EXAMINATION COMMITTEE:  
Kaloxylos Alexandros, Associate Professor  
Giorgos Tsoulos, Professor

1 November 2021

## ΠΕΡΙΛΗΨΗ

Στη σημερινή εποχή παρατηρείται ραγδαία, αύξηση χρήσης της τεχνολογίας, η οποία αδιαμφισβήτητα έχει μπει στις ζωές όλων, με τη μορφή κινητών συσκευών, υπολογιστών, καθώς και ποικίλων εφαρμογών, με λογικό επόμενο τη συνεχή αύξηση των απαιτήσεων αυτής. Ο κόσμος μέρα με την μέρα θέλει περισσότερα, όμως αποτελεί γεγονός πως η τωρινή τεχνολογία δικτύωσης, δυσκολεύεται να συμβαδίσει με τα θέλω αυτά. Την λύση έρχεται να δώσει ένα νέα μοντέλο δικτύωσης, το οποίο παίρνει τον έλεγχο από το υλικό και παρέχει την δυνατότητα όλες οι αποφάσεις ελέγχου να λαμβάνονται κεντρικά. Στο μοντέλο αυτό, η δικτύωση βασίζεται στο λογισμικό (Software Defined Networking- SDN) και ένα κεντρικό σημείο ελέγχου είναι αυτό, που παίρνει τις απαραίτητες αποφάσεις και διαχειρίζεται τον έλεγχο ροής, πορκειμένου να υπάρχει καλύτερη διαχείριση δικτύου και απόδοση εφαρμογών.

Στην πτυχιακή αυτή, παρουσιάζεται αρχικά η τεχνολογία SDN, και έπειτα με τη χρήση προγραμμάτων, όπως το Mininet και το D-ITG, δημιουργούνται εικονικά δίκτυα και παρατηρείται η απόδοση τους αλλάζοντας μεταβλητά στοιχεία του δικτύου.

Η δομή της πτυχιακής είναι η εξής: Στο 1ο κεφάλαιο παρουσιάζεται η παραδοσιακή αρχιτεκτονική δικτύων σε αντιδιαστολή με τη σύγχρονη. Στο 2ο κεφάλαιο υπάρχει αναλυτική περιγραφή της SDN τεχνολογίας ως προς την αρχιτεκτονική, το κύριο πρωτόκολλο που χρησιμοποιεί, τα σενάρια χρήσης της, καθώς και την προτυποποίηση, τα πλεονεκτήματα και τα μειονεκτήματά της. Έπειτα στο 3ο κεφάλαιο περιγράφεται αναλυτικά ο ελεγκτής SDN που χρησιμοποιήθηκε. Στο 4ο κεφάλαιο έχουμε παρουσίαση της “γεννήτριας κίνησης”, που χρησιμοποιήθηκε στα πειράματα της πτυχιακής. Το 5ο κεφάλαιο έχει την αναλυτική περιγραφή, του προγράμματος μέσω του οποίου δημιουργήθηκαν τα δίκτυα των πειραμάτων. Στο 6ο κεφάλαιο περιγράφονται αναλυτικά τα απαιτούμενα εργαλεία και οδηγίες για το κατέβασμα των στοιχείων που χρησιμοποιούνται. Στο κεφάλαιο 7 παρουσιάζονται τα πειράματα που πραγματοποιήθηκαν για τον υπολογισμό των στοιχείων απόδοσης των δικτύων στην εκάστοτε περίπτωση. Τέλος στο 8ο κεφάλαιο συνοψίζονται τα συμπεράσματα της παρούσας πτυχιακής εργασίας.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ:** Δίκτυα Επικοινωνιών

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:** Δικτύωση Βασισμένη στο Λογισμικό, Παρακολούθηση λειτουργίας, Δημιουργία κίνησης, SDN Controller, POX, OpenFlow, Mininet, D-ITG

## **ABSTRACT**

In this era, there is a rapid increase in the use of technology, which has undoubtedly entered everyone's lives in the form of mobile devices, computers, and various applications, with a logical consequence, the continuous increase in its demands. The world wants more and more day by day, but the fact is that the current networking technology is struggling to keep up with these wants. The solution comes from a new networking model, which takes control away from the hardware and enables all control decisions to be made centrally. In this model, networking is software based (Software Defined Networking- SDN) and it is a central point of control that makes the necessary decisions and manages flow control to provide better network management and application performance.

In this thesis, SDN technology is firstly introduced, and then through the use of programs such as Mininet and D-ITG, virtual networks are created and their performance is observed while changing variable network elements.

The structure of the thesis is as follows: Chapter 1 presents the traditional architecture of communication networks in contrast to the modern one. In chapter 2 there is a detailed description of SDN technology in terms of its architecture, the main protocol it uses, its usage scenarios, as well as its standardisation, advantages and disadvantages. Then in chapter 3 the SDN controller used is described in detail. In chapter 4 we have a presentation of the " traffic generator" used in the experiments of the thesis. Chapter 5 has the detailed description, of the program through which the networks of the experiments were created. Chapter 6 describes in detail the required tools and instructions for downloading the elements used. Chapter 7 presents the experiments conducted to calculate the performance data of the networks in each case. Finally, Chapter 8 summarises the conclusion of this thesis.

**SUBJECT AREA:** Communication Networks

**KEYWORDS::** Software Defined Networks, Monitoring, Traffic generation, SDN Controller, POX, OpenFlow, Mininet, D-ITG

This Thesis is dedicated to my father

## Contents

1. INTRODUCTION	11
1.1 Traditional Networking	11
1.2 Modern Networking	11
2. SOFTWARE-DEFINED NETWORKING(SDN)	13
2.1 Definition of SDN	13
2.2 SDN Architecture	13
2.3 OpenFlow Protocol	15
2.3.1 OpenFlow Switch	16
2.3.2 Group Table	16
2.3.3 Flow Table	16
2.3.4 WorkFlow	18
2.3.5 OpenFlow Protocol Messages	19
3. POX CONTROLLER	20
3.1 OpenFlow in POX	21
3.2 OpenFlow Statistics Events in POX	22
3.2.1 OpenFlow Modify State Messages	22
3.3 Pox Stock Components	22
4. D-ITG	23
4.1 Architecture	24
4.1.1 ITGSend	24
4.1.2 ITGRecv	25
4.1.3 ITGLog	25
4.1.4 ITGDec	25
4.2 Features	26
5. Mininet & Mininet-wifi	27
5.1 Mininet topologies	27
5.1.1 Simple default cmd-command topologies	27
5.1.2 Custom scripted topologies	28
5.1.3 Setting performance parameters	29
5.2 Miniedit	29
5.3 Mininet-wifi	30
6. Environment Set-up	31
6.1 System	31
6.2 Mininet download	31
6.3 Mininet-wifi download	35
6.4 D-ITG download	36
7. Experiments in Traffic Generation	38
7.1 First Experiment: 6-18 Host Topology	38
7.2 Second Experiment: Multiple Flows Networks	47
7.3 Third Experiment: Multiple Flows with Loss Networks	61
8. Conclusion	69
Appendix	70
References	74



## LIST OF FIGURES

Figure 1.1 Traditional and SDN architecture	12
Figure 2.1: SDN ARCHITECTURE	13
Figure 2.2 General OpenFlow design	15
Figure 2.3 OpenFlow Switch	16
Figure 2.4 Main components of a flow entry in a flow table	17
Figure 2.5 Packet flow in an OpenFlow switch	19
Figure 4.1 Architecture of D-ITG	24

## LIST OF TABLES

Table 1.1 Traditional networks and SDN comparison	12
Table 2.1: Flow entry parameters matching to flow entry components	18
Table 3.1 Comparison of the most popular open source SDN controllers	21
Table 7.1 Average Metrics	45

## LIST OF IMAGES

Image 5.1 Exemplary python script	28
Image 5.2 Miniedit example	29
Image 6.1 Mininet installation	31
Image 6.2 Clean-up	32
Image 6.3 Git installation	32
Image 6.4 Clone git	33
Image 6.5 Mininet version	33
Image 6.6 Full Mininet installation	34
Image 6.7 Test Mininet installation	34
Image 6.8 Mininet-wifi complete installation	35
Image 6.9 Test Mininet-wifi installation	35
Image 6.10 Unzip download	36
Image 6.11 G++ download	36
Image 6.12 Get D-ITG zipped source code	36
Image 6.13 Unzip source file	37
Image 6.14 Make	37
Image 6.15 Successful installation message	37
Image 7.1 Six Host topology and Pox activation(1st exp./pt1)	38
Image 7.2 D-ITG Simple traffic generation(1st exp./pt1)	39
Image 7.3 dat File(1st exp./pt1)	40
Image 7.4 Matlab script(1st exp./pt.1)	40
Image 7.5 Metrics' diagrams(1st exp./pt1)	41
Image 7.6 Eighteen Host topology and Pox activation(1st exp./pt2)	42

Image 7.7 D-ITG Simple traffic generation(1st exp./pt2)	42
Image 7.8 dat File(1st exp./pt.2)	43
Image 7.9 Matlab script(1st exp./pt.2)	43
Image 7.10 Metrics' diagrams(1st exp./pt.2)	44
Image 7.11 Mininet exit and cleanup	45
Image 7.12 Turn POX down	46
Image 7.13 Six Host topology and Pox activation(2nd exp./pt.1)	47
Image 7.14 Custom-Flow script (2nd exp./pt.1.1)	47
Image 7.15 D-ITG Multi-flow traffic generation(2nd exp./pt1.1)	48
Image 7.16 dat Files(2nd exp./pt.1.1)	49
Image 7.17 Matlab script(2nd exp./pt.1.1.1)	49
Image 7.18 Metrics' diagrams(2nd exp./pt.1.1.1)	50
Image 7.19 Matlab script(2nd exp./pt.1.1.2)	50
Image 7.20 Metrics' diagrams(2nd exp./pt.1.1.2)	51
Image 7.21 Custom-Flow script (2nd exp./pt.1.2)	51
Image 7.22 D-ITG Multi-flow traffic generation(2nd exp./pt1.2)	52
Image 7.23 dat Files(2nd exp./pt.1.2)	53
Image 7.24 Matlab script(2nd exp./pt.1.2.1)	53
Image 7.25 Metrics' diagrams(2nd exp./pt.1.2.1)	54
Image 7.26 Matlab script(2nd exp./pt.1.2.2)	55
Image 7.27 Metrics' diagrams(2nd exp./pt.1.2.2)	55
Image 7.28 Ten Host topology and Pox activation(2nd exp./pt.2)	56
Image 7.29 Custom-Flow script (2nd exp./pt.2)	57
Image 7.30 D-ITG Multi-flow traffic generation(2nd exp./pt2)	58
Image 7.31 dat Files(2nd exp./pt.2)	59
Image 7.32 Matlab script(2nd exp./pt.2)	59
Image 7.33 Metrics' diagrams(2nd exp./pt.2)	60
Image 7.34 Ten Host topology and Pox activation(3rd exp./pt.1)	61
Image 7.35 D-ITG Multi-flow traffic generation(3rd exp./pt1)	63
Image 7.36 dat Files(3rd exp./pt.1)	63
Image 7.37 Matlab script(3rd exp./pt.1)	64
Image 7.38 Metrics' diagrams(3rd exp./pt.1)	64
Image 7.39 Ten Host topology and Pox activation(3rd exp./pt.2)	65
Image 7.40 D-ITG Multi-flow traffic generation(3rd exp./pt2)	67
Image 7.41 dat Files(3rd exp./pt.2)	67
Image 7.42 Matlab script(3rd exp./pt.2)	68
Image 7.43 Metrics' diagrams(3rd exp./pt.2)	68

## **1. INTRODUCTION**

Nowadays, technological needs are rising like never before. From everyday use to industrial one, the demands have skyrocketed. Accessibility, mobility and flexibility have become major factors that need to be addressed and their problematic service be immediately resolved. Technology and more specifically networking has been struggling to keep up and that is due to the sheer rigidity and inflexibility that service provider networks show, when it comes to service creation and delivery. Especially with the advent of the new era, that fifth generation mobile networks bring, the need to change the conventional networking architecture is beyond apparent.

### **1.1 Traditional Networking**

In traditional architecture, network functionality is mainly implemented in dedicated hardware, such as application-specific integrated circuits. The functions of this conventional networking are primarily implemented through the use of one or multiple switches, routers and/or application delivery controllers). Both forwarding and control functions are performed in a distributed fashion using static protocols, by routing devices. Each device has a local control plane and a local data plane, as well as, its own management planes. Due to the implementation of the network's functions as hardware constructs, its speed is boosted. However, the complexity of the process of establishing the network topology using a control plane that runs locally is really high and thus time and money consuming making it prohibitive. Each and every device has to be connected to its data plane individually to make configuration changes or updates. The handling of network traffic is decided by the control plane, while its commands are executed in the data plane. This conventional networking, has now been outdated due to its inability to meet the dynamic computing and storage needs of today's changing business wants. It is due to its adherence to dedicated hardware, its rigidity that it can not perform real world experiments on large scale production networks. It becomes apparent that with its static nature it cannot pull through and meet the demands of today's networking.[1]

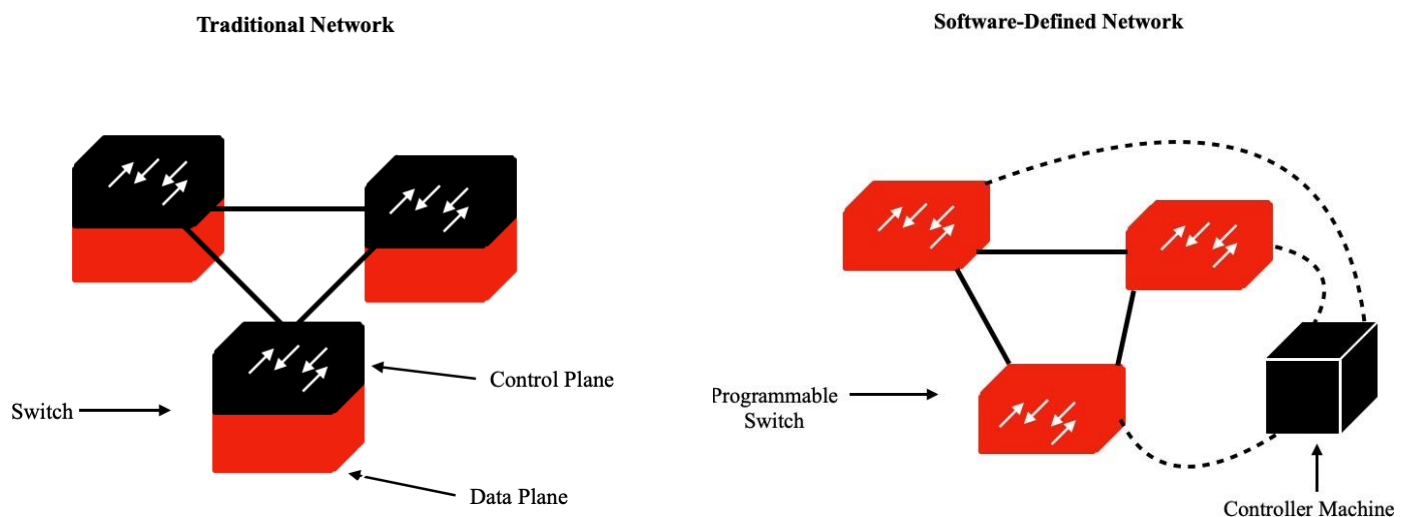
### **1.2 Modern Networking**

At the current time networking cannot live with the cons of traditional architecture and thus a new paradigm emerges, Software Defined Networking. SDN centralises and simplifies the control of enterprise network management, as well as offering traffic compatibility, agility, the ability to implement network automation, plus the capacity to generate policy-driven network supervision. Another pro of SDN, is that it lowers the operating costs, having an efficient administration, server utilisation improvements, and improved virtualisation control. In addition to that, it helps economically even more by "reviving" older network devices and simplifying the process of optimising commoditised hardware. Moreover, because of its cloud abstraction, SDN controllers are able to manage all the networking components that comprise the massive data center platforms. Lastly, another major advantage that modern networking has, is its ability to manipulate data traffic. This provides consistent and timely content delivery, whether it be for Voice over Internet Protocol or multimedia transmissions.[1]

In table 1.1 traditional and SDN networking is compared and in figure 1.1 you can see the difference in structure between the two architectures.

Criteria	Traditional Network	SDN
Network management	Difficult because changes are implemented separately at each device	Easier with the help of controller(s)
Global network view	Difficult	Central view at controller
Maintenance cost	Higher	Less
Time required for update/error handling	Sometimes it takes month	Quite easy because of central controller(s)
Controller utilisation	Not relevant	Important
Authenticity, Integrity and consistency of controller(s)	Not important	Important
Integrity and consistency of forwarding tables and network state	Important	Important
Availability of controller	Not relevant	Important
Resource utilisation	Less	High

**Table 1.1 Traditional networks and SDN comparison[2]**



**Figure 1.1 Traditional and SDN architecture**

## 2. SOFTWARE-DEFINED NETWORKING (SDN)

### 2.1 Definition of SDN

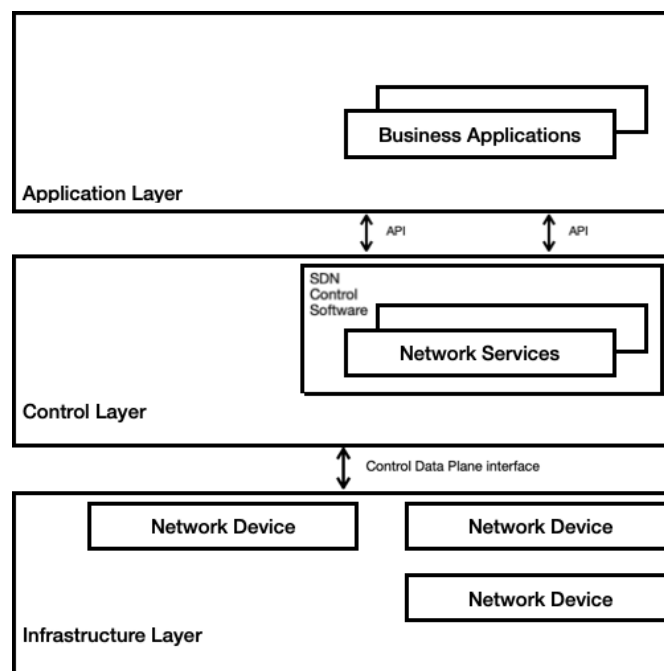
Open Networking Foundation (ONF) defines SDN as “an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.” [4] SDN is a game-changing architecture through which the control, of not only one networking device, but an entire network can be achieved. It comprises three layers: the application layer, the control layer and the infrastructure layer. It separates the network’s control logic from the hardware (i.e. routers and switches) and uses software-based controllers or application programming interfaces (APIs) in order to communicate with underlying hardware infrastructure and direct traffic on a network.

### 2.2 SDN Architecture

The three layers in an Software Defined-Network architecture are:

- Application: applications and services running on the network
- Control: SDN controller
- Infrastructure: switches and routers, and supporting physical hardware

In order for communication to exist between these layers, SDN uses northbound and southbound application program interfaces (APIs), where the northbound API enables communication between the application and the control layers, while the southbound API communicates between the infrastructure and control ones. [5]



**Figure 2.1: SDN ARCHITECTURE**

**Infrastructure layer:** It is the base layer, which contains physical, as well as virtual network devices, like routers and switches. As shown in Figure 2.1 the OpenFlow protocol, which will be defined in subchapter 2.3, is implemented in order for traffic forwarding rules to be put into effect.

**Control layer:** This layer acts as the brain for the entirety of the software defined network, being the centralised control plane, through the use of SDN controller(s), applications that control the traffic flows and “govern” the network. They will be described in chapter 3. In this layer, OpenFlow is also use, as the communication medium, to the infrastructure layer.

**Application layer:** The application layer contains typical network applications that organisations use. This could cover the likes of intrusion detection systems, load balancing etc. Whereas a tradition(non SD) network, would use an appliance specialised for that exact application(e.g firewall, load balancer), a SDN replaces the appliance with an application that uses the northbound API to communicate to the controller(s) in the control plane in order to manage the system’s behaviour.

**API:** The layers communicate with each other, using northbound and southbound APIs respectively. Infrastructure and Control layers through the Southbound-API(SBI) and Application and Control with the NorthBound-API(NBI). To get more into detail, SBI enables the definition of the switches’ and routers’s behaviour by the controller, while the NBI provides an abstraction of the network, thus leveraging network services and capabilities without being held tied to the specific details of their implementation.

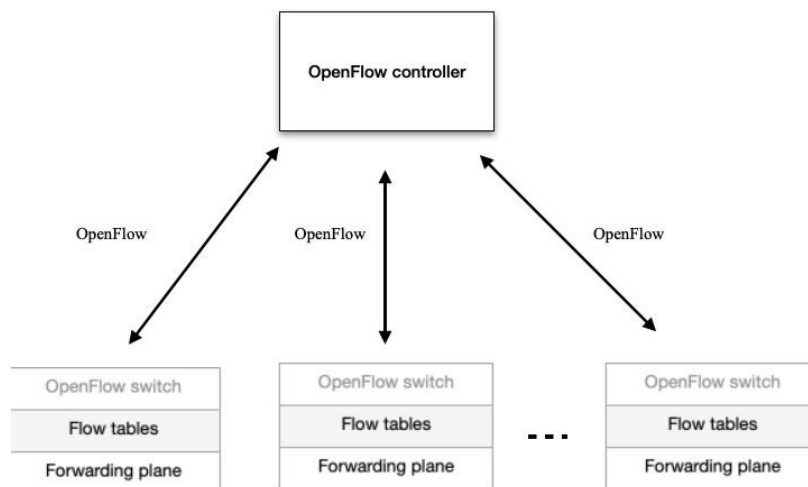
The most noteworthy SBI, is OpenFlow. OpenFlow’s main function is the enablement of the communication between the SDN controller and the network nodes in order for the network topology to be discovered by the router, to define network flows and implement requests relayed to it through the northbound APIs. Controllers use protocols such as OpenFlow to configure network devices. An OpenFlow switch has one or more tables of packet-handling rules (flow tables). Each rule matches a subset of the traffic and acts correspondingly(dropping, forwarding, etc.) on the traffic. Depending on the rules installed by a SDN controller application, an OpenFlow switch can act as a router, switch, firewall, or perform other roles (e.g., load balancer, traffic shaper, and in general those of a middle box) [6][7]

## 2.3 OpenFlow Protocol

OpenFlow (OF) is one of the first software-defined networking (SDN) standards. It defines the communication protocol in SDN architectures that enabled the SDN controller to directly interact with the forwarding plane of network devices such as switches and routers, physical, as well as virtual, so it can adapt to the ever changing enterprise requirements. This protocol uniquely identifies OpenFlow technology.

The protocol achieves communication through a set of messages, that are sent from the controller to the switch and a corresponding set of messages that are being sent in the opposite direction. This happens via a secure channel. Something that is made because OpenFlow protocol works on TCP. To get more into detail , the standard protocol is TCP, with 6633 being the controller's default port for OF V1.0 and 6653 for OF V1.3+. There also needs IP connectivity to be established between the controller and the switches for an OF connection to exist. Lastly, an OF channel is formed only after a successful TCP 3-way handshake takes place. It also has to be noted, that TLS can also be used instead of a TCP. In this instance, the controller and switch must have the proper certificates and keys for a successful TLS connection. This prevents any kind of snooping on the OF channel.

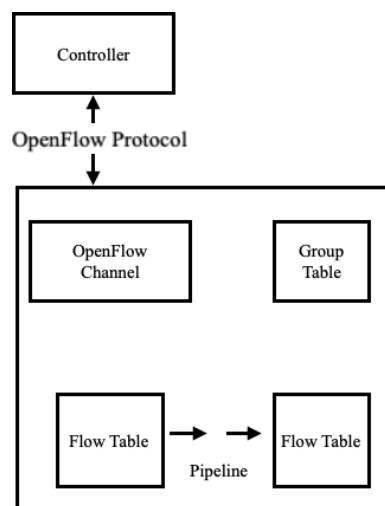
The messages, collectively, allow the controller to program the switch so as to allow control over the switching of user traffic. With the aid of some basic programming, one can define, modify and deletes flows. A set of rules specifies the forwarding actions that the device in question should take for every single packet belonging to that flow. When the controller defines a flow, it is providing the switch with the necessary information that is needed in order for it to treat incoming packets that match that flow. Truth is that the possibilities for treatment have grown more and more complex as the OpenFlow protocol has evolved, but the most basic ways of treatment for an incoming packet are to either forward the packet out one or more output ports, drop the packet, or pass the packet to the controller for exception handling. [8]



**Figure 2.2 General OpenFlow design**

### 2.3.1 OpenFlow Switch

An OF switch is an OpenFlow-enabled data switch that communicates over OpenFlow channel to an external controller. The switch communicates with the controller and the latter manages the switch via the OF switch protocol. It comprises one or more flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. OpenFlow switches are either based on the OpenFlow protocol or at-least compatible with it. An OF switch can only function with the “participation” of three essential elements: flow tables installed on switches, a controller and a proprietary OpenFlow protocol for the controller to talk securely with switches. Flow tables are set up on switches. Controllers dictate policies on flows to the switches via the OpenFlow protocol. The controller is able to set up paths through the network optimised for specific characteristics, such as speed, fewest number of hops or reduced latency. Moreover, through the OF protocol, the controller can update, add and delete flow entries in flow tables reactively ,as well as, proactively.[9]



**Figure 2.3 OpenFlow Switch**

### 2.3.2 Group Table

The group table contains group entries and each one of them contains a list of action buckets with specific semantics dependent on group type. The actions in one or more action buckets are applied to packets sent to the group. Switch designers have full freedom to implement the internals however they see convenient, provided that correct match and instruction semantics are preserved.

### 2.3.3 Flow Table

Each OF switch has its own flow table to define the paths of the packets. A flow table contains flow entries, and packets are matched based on the matching precedence of flow entries. Each one of them contains the fields shown in Figure 2.4:



Match Fields	Priority	Counters	Instructions	Timeout	Cookie
--------------	----------	----------	--------------	---------	--------

**Figure 2.4 Main components of a flow entry in a flow table**

- **Match Fields:** Minimally include ingress port and packet headers. Each packet is checked to see if it matches these fields. It can be wild carded (match any value) and in some cases bit masked.
- **Priority:** Checks the priority of the flow table entries.
- **Counters:** Updated when packets are matched
- **Instructions:** Instructions refer to the action for either pipeline processing or the action set (such as pop or push of a packet label).
- **Timeouts:** Maximum amount of time/ idle time before flow is expired by the switch.
- **Cookie:** This is used by the controller to filter flow entries.(Not used when processing packets) [8]

Each flow entry contains a set of specific instructions that are executed when a happens. These instructions lead to changes to the packet, action set and/or pipeline processing. It should be noted that, if a switch is unable to execute the instructions associated with a flow entry then it must reject it.

The most important instruction types are:

- **Apply-Actions action(s):** Applies the specific action(s) immediately, without causing any change to the Action Set. This instruction could be used to modify the packet between two tables or to execute multiple actions of the same type. The actions are specified as an action list.
- **Clear-Actions:** Clears all the actions in the action set right away.
- **Write-Actions action(s):** Merges the specified action(s) into the current action set. If an action of the given type exists in the current set, overwrite it, otherwise add it.
- **Goto-Table next-table-id:** Indicates the next table in the processing pipeline. The table-id has to be greater than the current table-id. The flow entries of the last table of the pipeline cannot include this instruction.

The flow tables of an OpenFlow switch are sequentially numbered, starting at 0. Pipeline processing always starts at the first flow table: the packet is first matched against flow entries of flow table 0.

An example of a flow entry is the following: cookie=0x0, duration=7.475s, table=0, n\_packets=0, n\_bytes=0, idle\_age=7, priority=00, in\_port=3 actions=output:1 where:

- **cookie:** Unique identifier of the flow
- **duration:** How long has the entry been in the flow table

- **n\_packets:** Number of packets matched by this field
- **n\_bytes:** Number of bytes matched by this field
- **priority:** Shows the order in which the specific flow will be examined
- **idle\_age:** How long has it been since a packet hit this rule
- **in\_port:** Is the match field of the flow. In this case, packets coming from port 3 are matched
- **actions:** Contains the actions that are going to be performed.
- **output:** The packet is forwarded to this port. [10]

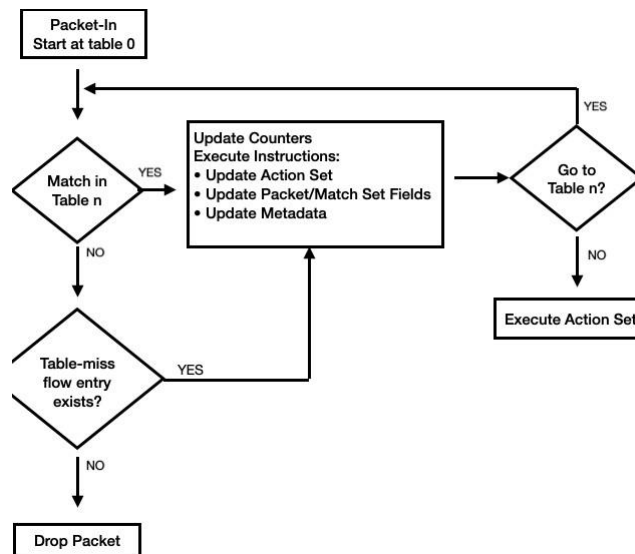
The mappings the above parameters to the components of the flow entry is shown in table 2.1

Match Fields	Priority	Counters	Instructions	Timeout	Cookie
in_port	priority	duration n_packets n_bytes	actions	idle_age	cookie

**Table 2.1: Flow entry parameters matching to flow entry components**

### 2.3.4 Workflow

The moment a user packet arrives at an OF switch, it is checked against an entry in a flow table to see if it matches; if there happens to be a match, then the packet is processed and forwarded according to the specified actions, sometimes visiting additional flow tables if also specified. If the packet does not match the last flow table entry, then it is dropped, if there is no table-miss entry. It has to be noted, that a table-miss entry can be considered as a default entry in order to let a packet know that if there is no match, then it is to be sent to the controller. If the match occurs only at the table-miss entry, then the packet is forwarded to the controller for further action. Then it is up to the controller to choose to either drop this packet, or insert a new flow table entry for this new flow. A representation of a packet flow in an OpenFlow switch can be seen in Figure 2.5. [8]



**Figure 2.5 Packet flow in an OpenFlow switch**

### 2.3.5 OpenFlow Protocol Messages

The OF Protocol supports three types of messages: controller-to-switch, asynchronous, and symmetric, each with several sub-types. Controller/switch messages are initiated by the controller, may or may not require a response from the switch and are used for managing or examining the switch state while asynchronous messages are sent without a controller, soliciting them from a switch and are used to inform and update the controller in case of network events and possible changes to the state of the switch. Symmetric messages are sent without solicitation, in either direction.

#### Controller-to-switch:

**Features:** This performed commonly upon establishment of the OpenFlow channel. A request is sent from the controller so as to be informed about the capabilities of a switch and the switch must reply with a feature's reply message that specifies the features and capabilities of the switch.

**Configuration:** The controller can set and query configuration parameters in the switch. The switch can only respond to a query from the controller.

**Modify-State:** These messages are sent by the controller in order to manage the state of the switches. The controller can add, delete or modify flow table entries by using this kind of messages or set switch port priorities.

**Read-State:** Read-State messages collect information, such as statistics, current configuration and capabilities from the switch as far as flow tables, ports and the flow entries are concerned.

**Packet-out:** These messages are used by the controller to send packets out of a specified port on the switch, and forward packets received by Packet-in messages. Packet-out messages have to contain a full packet or a buffer ID referencing a packet stored in the switch. The message has to contain a list of actions, that are to be applied in the order they are specified; an empty action list drops the packet.

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

**Role-Request:** This is a message mostly useful when the switch connects to multiple controllers. These messages are used by the controller in order to set or query the role of its OpenFlow channel.

#### **Asynchronous:**

**Packet-in:** This kind of messages assign the control of a packet to the controller. For all packets that do not have a matching flow entry or if a packet matches an entry with a send to controller action, a packet-in message is delivered to the controller. Packet-in events can be configured to buffer packets. If the switch has sufficient memory to buffer packets that are sent to the controller, the packet-in message contains some fraction of the packet header (the default is 28 bytes) and a buffer ID to be used by the controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering (or have run out of internal buffer space) have to send the entirety of the packet to the controller as part of the message.

**Flow-Removed:** These messages inform the controller about the removal of a flow entry from a flow table. FlowRemoved messages are only sent for flow entries that have the OFPFF\_SEND\_FLOW\_REM flag set. They are generated as the result of a controller flow delete requests or the switch flow expiry process when one of the flow timeout is exceeded.

**Port-status:** Inform the controller about a port change. The switch is expected to send port-status messages to controllers as port configuration or port state changes. These kinds of events include changes in port configuration events for example if it was brought down directly by a user, and port state change events, for instance if the link went down.

**Error:** The switch is able to send a notification message to the controllers in case of a problem occurrence.

#### **Symmetric:**

**Hello:** Messages exchanged between the switch and controller at the connection startup.

**Echo:** Echo request messages are initiated from either the switch or the controller and must respectively return an echo reply. They are mostly used to verify the liveness of a controller-switch connection and indicate latency or bandwidth.

**Experimenter:** These messages provide a definite way for OpenFlow switches to offer additional functionality within the OpenFlow message type space, in order to set up the ground for future OpenFlow revisions. [11]

### **3. POX CONTROLLER**

The most valuable piece in SDN is undoubtedly their brain, the SDN controller. It is the element that handles the intelligence of the network and therefore dictates the actions that need to be done, in order to ensure a fine network functionality. The immense growth of SDN has led to the development of numerous different controllers, each one with its pros and cons. An overview of the most popular SDN controllers and their features is shown in Figure 3.1. In this thesis, the POX controller will be used. POX is an OpenFlow controller, that is developed through the use of python. The controller comes pre-installed with Mininet, which will be addressed in detail later on. POX provides a framework for communicating with SDN switches by the use of either the OpenFlow or the OVSDB protocol. Due to its simplicity it is a popular tool for teaching about and researching software defined networks and network applications programming.

POX can be immediately used as a basic SDN controller through the use of the stock components that come bundled with it. [12]

	Program- ming Language	GUI	Docum- entation	Modularity	Distributed/ Centralized	Platform Support	Productivity	Southbound Apis	Northbound Apis	Partner	Multithreadin g Support	Openstack Support
<b>ONOS</b>	Java	Web Based	Good	High	D	Linux,MAC OS, And Windows	Fair	OF1.0, 1.3, NETCO NF	REST API	ON.LAB, At&T, Ciena,Cisco, Ericsson,Fujitsu, Huawei,Intel, Nec,Nsf.Ntt Communication, Sk Telecom	Y	N
<b>Open- Day- Light</b>	Java	Web Based	Very Good	High	D	Linux,MAC OS, And Windows	Fair	OF1.0, 1.3, 1.4, NET- CONF/ YANG, OVSDb, PCEP, BGP/LS, LISP, SNMP	REST API	Linux Foundation With Memberships Covering Over 40 Companies, Such As Cisco, IBM, NEC	Y	Y
<b>NOX</b>	C++	Python + QT4	Poor	Low	C	Most Supported On Linux	Fair	OF 1.0	REST API	Nicira	NOX_ MT	N
<b>POX</b>	Python	Python + QT4	Poor	Low	C	Linux,MAC OS, And Windows	High	OF 1.0	REST API	Nicira	N	N
<b>RYU</b>	Python	Yes	Fair	Fair	C	Most Supported On Linux	High	OF 1.0, 1.2, 1.3, 1.4, NETCO NF, OF- CONFIG	REST For South bound	Nippo Telegraph And Telephone Corporation	Y	Y

**Table 3.1 Comparison of the most popular open source SDN controllers**

### 3.1 OpenFlow in POX

As it could be expected, the OF protocol is implemented in POX. The communication it creates might go from the controller to a switch, or vice versa. When communication is from the controller to the switch, this is performed by controller code that sends an OF message to a particular switch. At a reversed situation, messages show up in POX as events for which a developer can write event handlers – an event type corresponding to each message type that a switch might send generally exists. This communication can be effectively attained via either a Connection object for that particular switch or via an OpenFlow Nexus, that responsible for managing this switch. Each switch has one Connection object connected to POX and one OpenFlow Nexus, which manages all connections. As far as the Connection is concerned, every time a connection happens between a switch and POX, there is an associated Connection object. Connection objects have a lot of functionalities, such as being sources of events from switches and many other attributes and sending commands to switches. On the other hand, OpenFlow nexus is available as core.openflow and can manage a set of OpenFlow Connections. Generally, both ways of communication can be used in an application, but it makes sense to use one or the other depending on the situation. [13]

## 3.2 OpenFlow Statistics Events in POX

Information about the link usage in a network, that is critical in order to determine if a congestion is possible and if it can be prevented, is collected through requests made by the controller for the switches. This situation is called statistic event, and more specifically it occurs when the controller receives an `OFT_STATS_REPLY`, which is sent from a switch in response to the controller's `OFPT_STATS_REQUEST`. Separate events are assigned to every separate statistic reply type. [13]

### 3.2.1 OpenFlow Modify State Messages

If one wants to modify flow-table entry from the controller, they can do it through the use of `OFPT_FLOW_MOD` message. This enables the controller to customise the flow entries after been synchronised by the switch. Since there was not any use of this in our experiments, we will not go further into detail. You can find more info specialised at this at the POX wiki. [14]

## 3.3 POX Stock Components

POX comes with a number of stock components. Components in POX are usually arguments that can be put on the POX command line to either provide us a kind of functionality, or even some convenient features. This is called “Invoking POX”.

POX components are extra Python programs and some of the most important are the ones listed below:

**Py:** This component causes POX to start an interactive Python interpreter that I used for debugging and interactive experimentation.

**forwarding.l2\_learning:** The L2 Learning component makes OpenFlow switches act as L2(Ethernet) learning switches. It learns Ethernet MAC addresses, and matches all fields in the packet header so it may install multiple flows in the network for each pair of MAC addresses. For instance, different TCP connections will ultimately lead to different flows being installed.

This L2 Learning component keeps flow tables manageable by creating flows with idle timeout values, so that switches will automatically delete flows that have not serviced packets during the timeout period (which is ten seconds).

**openflow.discovery:** This component uses special LLDP messages sent to and received from OpenFlow switches in order to determine the network topology. It also detects and thus raises events, when network links go up or down.

**openflow.spanning\_tree --no-flood --hold-down:** This component is required in cases where the topology of the network contains loops. It “cooperates” with the OpenFlow Discovery component to build a view of the network topology and constructs a spanning tree by disabling flooding on switch ports that aren't on the tree. The options `no-flood` and `hold-down` are used in order to make sure no packets are flooded in the network before the component creates the spanning tree.

The Spanning Tree component will respond to changes in the network topology. If a link is broken, and if an alternate link exists, connectivity can be maintained in a network by creating a new tree that enables flooding on the ports connected to the alternate link.

It is best house a forwarding component in which the created flows have a set timeout value, when using the Spanning Tree component.

**host\_tracker:** The Host Tracker component attempts to keep track of hosts in the network. If a change occurs, the component raises a HostEvent. In brief, host\_tracker works by examining packet-in messages and learning MAC and IP bindings. We then systematically ARP-ping hosts to see if they're still there. This means it relies on packets coming to the controller, so forwarding in the network must be done reactively. In accordance with that, we have to use a forwarding component like forwarding.l2\_learning.

**info.packet\_dump:** This component will display on the log console information about data packets received from switches. It is quite similar to the tcpdump switch command.

**log.level --DEBUG:** The Log Level component allows the user to specify the amount of detail they wish to see in the log information produced by POX. The final and most detailed level is DEBUG.

**samples.pretty\_log:** This component formats log messages into a custom log format to in order to produce an attractive and readable log output on the POX console. [15]

#### 4. D-ITG

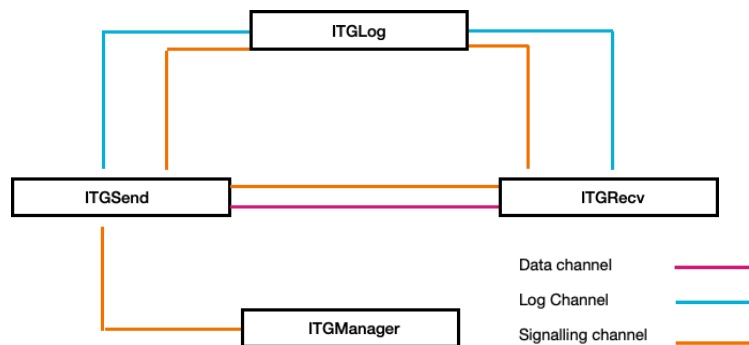
D-ITG (Distributed Internet Traffic Generator) is a platform with the ability to create IPv4, as well as, IPv6 traffic by accurately replicating the workload of current Internet applications. Meanwhile D-ITG is also a network measurement tool capable of measuring the common performance metrics (e.g. throughput, delay, jitter, packet loss) at packet level.

D-ITG is able to generate traffic, that accurately adheres to patterns that are defined by the inter departure time between packets (IDT) and the packet size (PS) stochastic processes. Through the specification of the distributions of IDT and PS random variables, the generator managed to emulate various protocols, such as: TCP, UDP, ICMP, DNS, Telnet and VoIP (G.711, G.723, G.729, Voice Activity Detection, Compressed RTP). At the transport layer, D-ITG currently supports TCP (Transmission Control Protocol), UDP (User Datagram Protocol), SCTP1 (Stream Control Transmission Protocol), and DCCP1 (Datagram Congestion Control Protocol), as well as, ICMP (Internet Control Message Protocol). FTP-like passive mode is also supported to conduct experiments in presence of NATs, and it is also possible to set the TOS (DS) and TTL IP header fields.

[16][17]

## 4.1 Architecture

D-ITG has a distributed multi-component architecture. In Figure 4.1 a graphical overview on the relationship among the main bricks of D-ITG platform, is shown. The communication between sender and receiver is done through the use of a separate signalling channel and is ruled by a protocol for the configuration of the experiment (Traffic Specification Protocol - TSP).[17]



**Figure 4.1 Architecture of D-ITG**

### 4.1.1 ITGSend

The ITGSend component is the one responsible for the generation of traffic flows and works in three modes:

**Single-flow mode:** The single-flow mode enables ITGSend to generate one traffic flow according to the given command-line options. The flow is managed by a dedicated thread, while a separate thread is responsible for the set up and coordination of the generation process through communication with the ITGRecv on a separate channel.

**Multi-flow mode:** The multi-flow mode enables ITGSend to simultaneously generate several flows. Each flow is managed by a single thread, with a single, separate thread acting as a master and coordinating the others. In order to generate a x number of flows, x lines must be contained in the script file, each of them used to specify the characteristics of one flow. Each line can contain all the options shown below, with the exception of those regarding the logging process. When using this mode, the logging options must be specified on the command line and they refer to every flow.

**Daemon mode:** The daemon mode allows ITGSend to be remotely controlled by using the ITGapi(C++ API which will not be shown in this thesis). When working in this mode ITGSend acts as a daemon listening on a UDP port for traffic generation requests.

In order to use ITGSend in each mode you have to use these commands in the cmd:

Single-flow mode:

```
$ ./ITGSend [log_opts] [sig_opts] [flow_opts] [misc_opts] [ [idt_opts] [ps_opts] | [app_opts] ]
```

• Multi-flow mode:

```
$ ./ITGSend [log_opts]
```



- Daemon mode:  
\$ ./ITGSend -Q [log\_opts]

#### 4.1.2 ITGRecv

The ITGRecv component is responsible for receiving multiple parallel traffic flows generated by one or more ITGSend instances. It normally runs as a multi-threaded daemon that listens to a TCP socket for incoming traffic reception requests. A new thread is created every time a request is received from the network, and it performs all the operations related to the new request (e.g. receiving the packets of the flow). The port numbers on which ITGRecv will receive each flow and any logging activity required on the receiver side can be remotely controlled by ITGSend. TSP signaling protocol, allows ITGRecv and ITGSend to properly setup and manage the traffic generation process.

In order to use ITGRecv you have to use this command in the cmd:

```
./ITGRecv [options]
```

#### 4.1.3 ITGLog

The ITGLog component's role is the receiving and storing of log information possibly sent by ITGSend and ITGRecv. It runs as a multi-threaded daemon listening on a TCP socket for incoming log requests. Log information is received over TCP or UDP protocols on port numbers dynamically allocated in the range [9003–10003].

In order to use ITGLog you have to use this command in the cmd:

```
./ITGLog [options]
```

#### 4.1.4 ITGDec

The role of the ITGDec component is to decode and analyse the log files stored during the experiments conducted by using D-ITG. ITGDec parses the log files generated by ITGSend and ITGRecv and calculates the average values of bitrate, delay and jitter either on the whole duration of the experiment or on variable-sized time intervals. ITGDec analyses the log files produced by ITGSend, ITGRecv, and ITGLog in order to produce the following results about each flow and about the whole set of flows:

- Synthetic reports:
  - Experiment duration
  - Packets transferred
  - Payload bytes transferred
  - One-way/round-trip delay (minimum, maximum, average, standard deviation)
  - Average bitrate – Average packet rate
  - Dropped packets
  - Duplicate packets

- Loss events
- Average loss-burst size
- First/last sequence number

- Sampled QoS metrics timeseries:

- Bitrate [Kbps] (i.e. goodput)
- One-way/round-trip delay [ms]
- Jitter [ms] (i.e. delay variation)
- Packet loss [pps] (i.e. packets lost per second)

In order to use ITGDec you have to use this command in the cmd:

```
./ITGDec <logfile> [options]
```

The most important options, for the options part of each component are shown in the Appendix. For even more info check the D-ITG documentation. x+12

## 4.2 Features

D-ITG is able to generate multiple unidirectional flows from many senders toward many receivers, each of them having the following features.

- Customisable flow-level properties
  - duration
  - start delay
  - total number of packets
  - total number of Kbytes
- Supported Layer-3 features
  - protocols: IPv4, IPv6
  - customisable header fields:
    - \* source and destination IP addresses
    - \* source interface binding (for multi-homed devices)
    - \* initial TTL value
    - \* DS byte
  - NAT traversal: FTP-like passive mode
- Supported Layer-4 features
  - protocols: TCP, UDP, ICMP, DCCP, SCTP
  - customizable header fields:
    - \* source and destination port numbers
- Supported Layer-7 features
  - Predefined stochastic PS (Packet Size) and IDT (Inter Departure Time) profiles:
    - \* Telnet
    - \* DNS
    - \* Quake3

- \* CounterStrike (active and inactive)
- \* VoIP (G.711, G.729, G.723)
- Payload content: random or read from file
- Stochastic processes supported for both PS and IDT:
  - \* Supported distributions are Uniform, Constant, Exponential, Pareto, Cauchy, Normal, Poisson, Gamma, Weibull
  - \* Explicit random seed selection for replicating the same stochastic process
  - \* Loading of PS and IDT series from file 5
- Packet-level QoS metrics
  - Bitrate
  - Packet rate
  - One way delay (requires clocks synchronisation)
  - Round Trip Time
  - Jitter
  - Packet loss

[16]

## 5. Mininet & Mininet-wifi

Mininet is a network emulator that is used to create networks comprised of virtual hosts, switches, controllers, and links. The realistic virtual network that it generates, runs on a single machine and the user can have an easy time interacting with it, through the use of Command Line Interface (CLI). The behaviour of Mininet's virtual hosts, switches, links and controller is akin to discrete hardware elements. Very often, it is possible to create a Mininet network that resembles a hardware network or vice versa and to run even the same binary code and applications on either platform.

[18]

### 5.1 Mininet topologies

In Mininet one can easily create a network setting the different parameters they desire.

#### 5.1.1 Simple default cmd-command topologies

By typing: “\$ sudo mn” or “\$ sudo mn --topo=minimal” at the cmd you gain access to Mininet's default topology. It consists of one OpenFlow switch, two hosts connected to it and the OpenFlow reference controller.

The topology used can be easily changed to other basic form, just by “expanding” the previous command.

For instance : “sudo mn --topo single,[HOST\_NUMBER]”

The argument --topo followed by a parameter is added to change this topology to a different one.

Here, the “single” parameter indicates a topology that has a single switch and HOST\_NUMBER of hosts connected to it.

Another example would be the use of “\$ sudo mn --topo linear,[SWITCH\_NUMBER]” command which creates a linear topology that has SWITCH\_NUMBER of switches, where each switch has one host and all switches connect in a line.

### 5.1.2 Custom scripted topologies

Another way of changing the topology of your virtual network aside from simple commands like these, is through the use of a simple Python API. The python script is passed as a parameter to Mininet with a command, like this:

```
“$ sudo mn -- custom [PYTHON_SCRIPT_PATH] --topo mytopo”
```

```
1 from mininet.topo import
2
3 class MyTopo ( Topo ): "Simple topology example."
4
5 def _init( self ):
6
7     "Create custom topo."
8
9     # Initialize topology
10    Topo._init__(self)
11
12    # Add hosts and switches
13    leftHost = self.addHost ( 'h1' )
14
15    rightHost =self.addHost('h2' )
16
17    leftswitch = self.addswitch('s3' )
18
19    right Switch = self.addSwitch('s4')
20
21    #Add links
22    self.addlink ( leftHost, leftswitch)
23
24    self.addLink ( leftswitch, right Switch)
25
26    self.addlink ( rightSwitch, rightHost )
27
28
29    topos "nytopo : lambda: MyTop ( )
30
```

Image 5.1 Exemplary python script

Some of the most important classes, methods, functions and variables that are used in a script topology are the following:

**Topo:** The base class for Mininet topologies

**Init():** One of the reserved methods in Python. It is called as a constructor. It is called when an object is created from a class and access is required to initialise the attributes of the class.

**addSwitch():** Adds a switch to a topology and returns the switch name

**addHost():** Adds a host to a topology and returns the switch name

**addLink():** Adds a bidirectional link to a topology

**Mininet():** Main class to create and manage a network

**start():** Starts the network

**dumpNodeConnections():** Dumps connections to/from a set of nodes

**setMac():** Set the MAC address for a host or specific interface

**CLI():** Command-line interface which provides a variety of useful commands as well as the ability to display xterm windows and run commands on individual hosts in the network. CLI (net) is used to pass the network object into the CLI constructor. Useful for debugging the network, as it allows to have a view of the network topology using the net command, test the connectivity in the network using the pingall command and send commands to individual hosts.

**stop():** stops the network

### 5.1.3 Setting performance parameters

Mininet also allows you to set specific performance parameters like delay, bandwidth, loss, etc. In order to create links with bandwidth/delay support, we set `Link=TCLink` in the `Mininet()` call in `main()`. The `TCLink` class represents a Traffic Controlled Link. An example of the method `addLink` (method used to specify parameters) is shown below:

```
self.addLink(node1, node2, bw=20, delay='3ms', max_queue_size=100, loss=9)
```

The above adds a bidirectional link with bandwidth, delay and loss characteristics, with a maximum queue size of 100 packets. The parameter `bw` is expressed as a number in Mbps, `delay` is expressed as a string with units in place (e.g. '5ms', '100us', '1s'), `loss` is expressed as a percentage (between 0 and 100) and `max_queue_size` is expressed in packets. One can also set link parameters automatically from the command line. For instance:

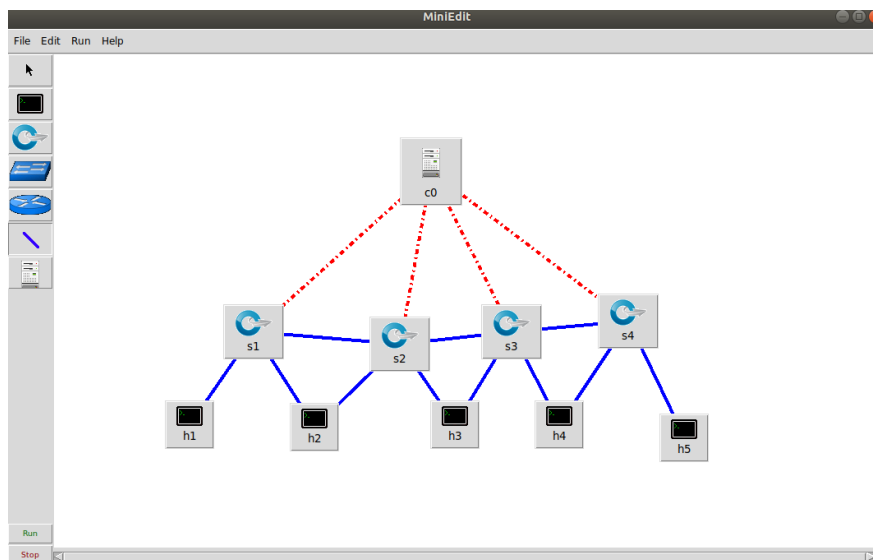
```
'$ sudo mn -link tc, bw=10, delay=10ms'
```

[19]

### 5.2 Miniedit

Most of the above functionalities can be provided by Miniedit, which is a simple GUI editor for Mininet. It is an experimental tool created to demonstrate how Mininet can be extended. To execute it you have to browse to the examples folder of Mininet and then type “`$ sudo python miniedit`”.

It is a very useful feature that allows the creation of custom topologies and the setting of parameters, without writing a single line of code. All its tools as well as an extensive guide on how to operate the editor is given by Brian Linkletter and is referenced below. [20]



**Image 5.2 Miniedit example**

### 5.3 Mininet-Wifi

In this thesis we chose to work with Mininet-Wifi. Mininet-WiFi is a fork of the Mininet SDN network emulator and extends the functionality of Mininet, through the addition of virtualised WiFi stations and access points based on the standard Linux wireless drivers and the 80211\_hwsim wireless simulation driver. Certain classes were added in order to support the addition of these wireless devices in a Mininet network scenario and to emulate the attributes of a mobile station such as position and movement relative to the access points. The Mininet-WiFi simply extended the base Mininet code by adding or modifying classes and scripts. Therefore, Mininet-WiFi adds new functionality while still supporting all the normal SDN emulation capabilities of the standard Mininet network emulator. [21]

The reason behind this choice, is that Mininet-Wifi obviously (being practically a newer edition) offers much more adaptability (adding wireless communication) and could possibly be the setting ground of future work. With this being said we will not be involved in any wifi-based experiments, so we will use it, as if we were using the original Mininet.

## 6. Environment Set-up

In this chapter the installation process, of all programs used in this thesis, is thoroughly described.

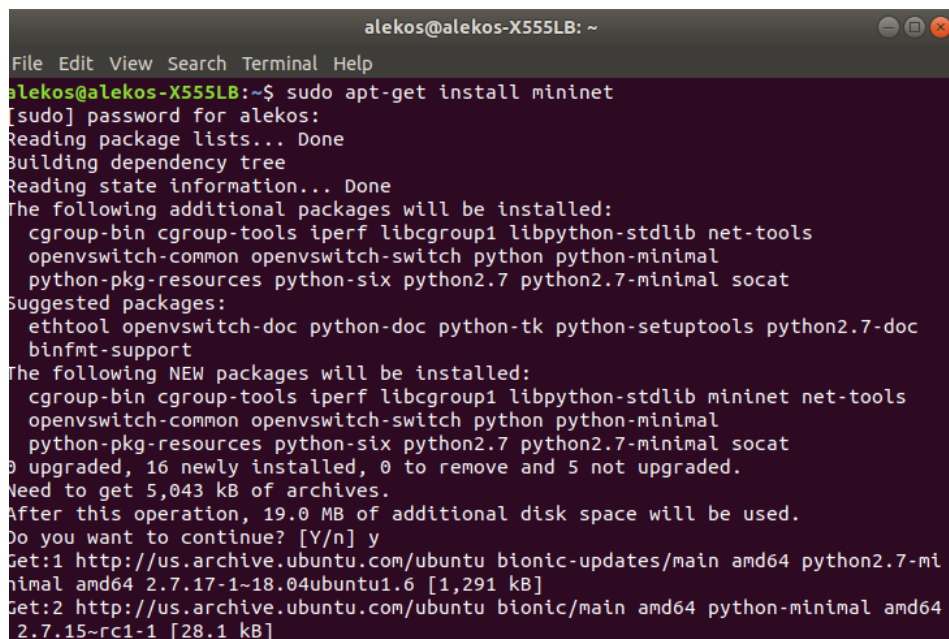
### 6.1 System

- Ubuntu 18.04 LTS
- Memory: 8GB
- Graphics: Intel® HD Graphics 5500 (BDW GT2)
- OS type: 64-bit
- Disk: 250 GB SSD

### 6.2 Mininet download

Open your terminal(shortcut cntr+alt+T) and type the following commands in order.

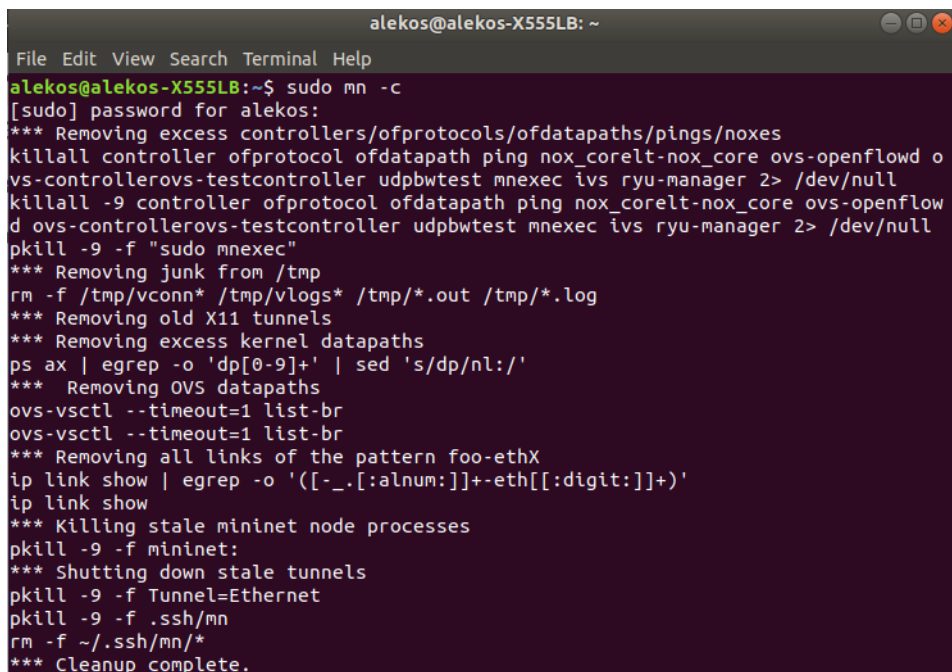
Firstly, type “sudo apt-get install mininet” and when the time comes press “y” in order for the installation to continue.



```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo apt-get install mininet  
[sudo] password for alekos:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  cgroup-bin cgroup-tools iperf libcgroup1 libpython-stdlib net-tools  
  openvswitch-common openvswitch-switch python python-minimal  
  python-pkg-resources python-six python2.7 python2.7-minimal socat  
Suggested packages:  
  ethtool openvswitch-doc python-doc python-tk python-setuptools python2.7-doc  
  binfmt-support  
The following NEW packages will be installed:  
  cgroup-bin cgroup-tools iperf libcgroup1 libpython-stdlib mininet net-tools  
  openvswitch-common openvswitch-switch python python-minimal  
  python-pkg-resources python-six python2.7 python2.7-minimal socat  
0 upgraded, 16 newly installed, 0 to remove and 5 not upgraded.  
Need to get 5,043 kB of archives.  
After this operation, 19.0 MB of additional disk space will be used.  
Do you want to continue? [Y/n] y  
Get:1 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 python2.7-mi  
nimal amd64 2.7.17-1~18.04ubuntu1.6 [1,291 kB]  
Get:2 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 python-minimal amd64  
2.7.15~rc1-1 [28.1 kB]
```

Image 6.1 Mininet installation

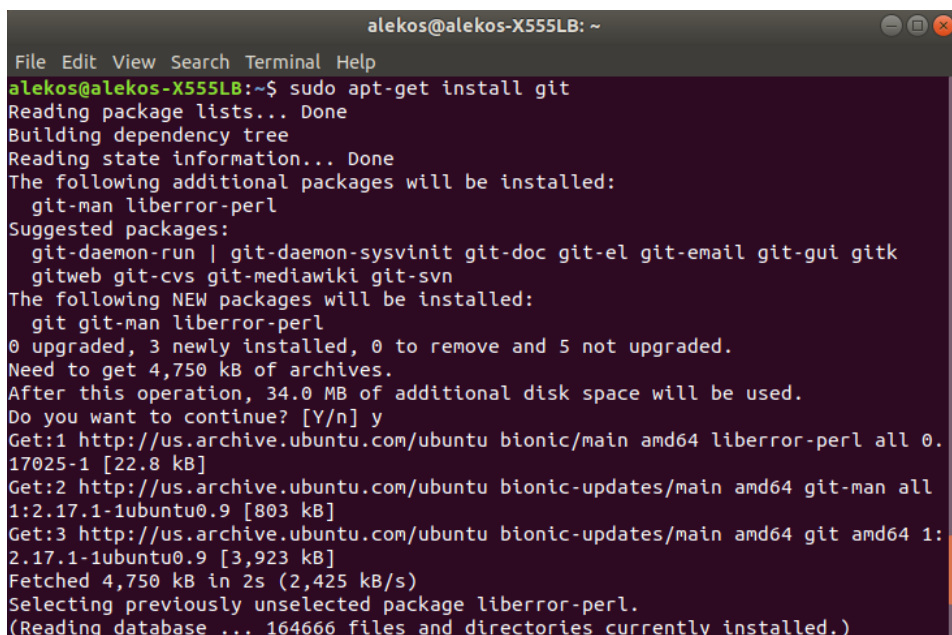
Then use “sudo mn” -c to clean-up



```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo mn -c  
[sudo] password for alekos:  
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes  
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflow o  
vs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null  
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflow  
d ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null  
pkill -9 -f "sudo mnexec"  
*** Removing junk from /tmp  
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log  
*** Removing old X11 tunnels  
*** Removing excess kernel datapaths  
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'  
*** Removing OVS datapaths  
ovs-vsctl --timeout=1 list-br  
ovs-vsctl --timeout=1 list-br  
*** Removing all links of the pattern foo-ethX  
ip link show | egrep -o '([-_.[:alnum:]]+-eth[:digit:]]+)'  
ip link show  
*** Killing stale mininet node processes  
pkill -9 -f mininet:  
*** Shutting down stale tunnels  
pkill -9 -f Tunnel=Ethernet  
pkill -9 -f .ssh/mn  
rm -f ~/.ssh/mn/*  
*** Cleanup complete.
```

Image 6.2 Clean-up

Then you install git with “sudo apt-get install git”.



```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo apt-get install git  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  git-man liberror-perl  
Suggested packages:  
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk  
  gitweb git-cvs git-mediawiki git-svn  
The following NEW packages will be installed:  
  git git-man liberror-perl  
0 upgraded, 3 newly installed, 0 to remove and 5 not upgraded.  
Need to get 4,750 kB of archives.  
After this operation, 34.0 MB of additional disk space will be used.  
Do you want to continue? [Y/n] y  
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 liberror-perl all 0.  
17025-1 [22.8 kB]  
Get:2 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git-man all  
1:2.17.1-1ubuntu0.9 [803 kB]  
Get:3 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git amd64 1:  
2.17.1-1ubuntu0.9 [3,923 kB]  
Fetched 4,750 kB in 2s (2,425 kB/s)  
Selecting previously unselected package liberror-perl.  
(Reading database ... 164666 files and directories currently installed.)
```

Image 6.3 Git installation



And after that type “git clone git://github.com/mininet/mininet” to clone mininet from its git repository.

```
alekos@alekos-X555LB:~$ git clone git://github.com/mininet/mininet
Cloning into 'mininet'...
remote: Enumerating objects: 10178, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 10178 (delta 8), reused 16 (delta 6), pack-reused 10154
Receiving objects: 100% (10178/10178), 3.21 MiB | 1.37 MiB/s, done.
Resolving deltas: 100% (6790/6790), done.
alekos@alekos-X555LB:~$
```

**Image 6.4 Clone git**

Type “cd mininet” and then “git tag” to see all the mininet versions and select one with the “git checkout -b \_\_\_\_” command. In this case, I chose mininet 2.3.0. Then use “cd ..”.

```
alekos@alekos-X555LB: ~
File Edit View Search Terminal Help
alekos@alekos-X555LB:~$ cd mininet
alekos@alekos-X555LB:~/mininet$ git tag
1.0.0
2.0.0
2.1.0
2.1.0p1
2.1.0p2
2.2.0
2.2.1
2.2.2
2.3.0
2.3.0b1
2.3.0b2
2.3.0d3
2.3.0d4
2.3.0d5
2.3.0d6
2.3.0rc1
2.3.0rc2
cs244-spring-2012-final
alekos@alekos-X555LB:~/mininet$ git checkout -b mininet-2.3.0 2.3.0
Switched to a new branch 'mininet-2.3.0'
alekos@alekos-X555LB:~/mininet$ cd ..
alekos@alekos-X555LB:~$
```

**Image 6.5 Mininet version**

After that type “mininet/util/install.sh -a” in order to install everything included in the mininet VM.

```
alekos@galekos-X555LB:~$ mininet/util/install.sh -a
Detected Linux distribution: Ubuntu 18.04 bionic amd64
sys.version_info(major=2, minor=7, micro=17, releaselevel='final', serial=0)
Detected Python (python) version 2
Installing all packages except for -eix (doxypy, ivs, nox-classic)...
Install Mininet-compatible kernel if necessary
Hit:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu bionic-security InRelease
Reading package lists... Done
Reading package lists...
Building dependency tree...
Reading state information...
linux-image-5.4.0-84-generic is already the newest version (5.4.0-84.94~18.04.1)
.
linux-image-5.4.0-84-generic set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
Installing Mininet dependencies
Reading package lists...
Building dependency tree...
Reading state information...
```

**Image 6.6 Full Mininet installation**

This will take a few minutes. After that is over, you can use the command “sudo mn”, to create mininet’s base topology to test if it is working.

```
alekos@galekos-X555LB:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

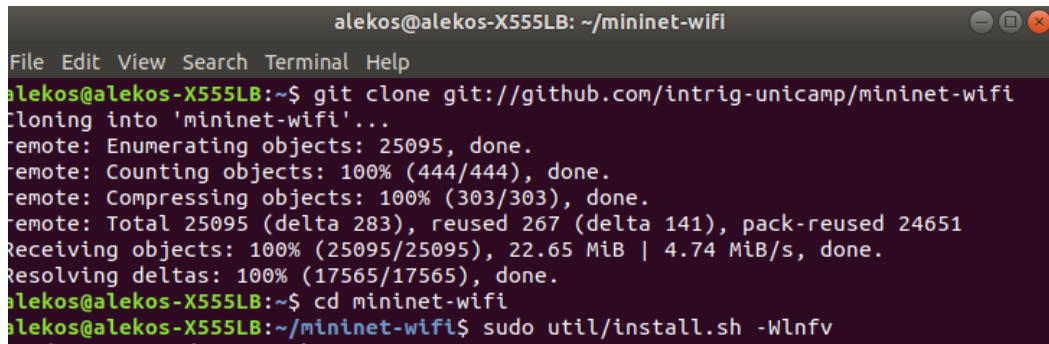
**Image 6.7 Test Mininet installation**

### 6.3 Mininet-wifi download

Open your terminal(shortcut cntr+alt+T) and type the following commands in order.

Type “git clone git://github.com/intrig-unicamp/mininet-wifi” to clone mininet-wifi from its git repository. \*\*If you have not already installed git check above at Image 6.3\*\*

Then use the command “cd mininet-wifi” and after that type “sudo util/install.sh -Wlnfv”, to install mininet-wifi with wireless dependencies, wmediumd, mininet-wifi dependencies, OpenFlow, OpenvSwitch. You could also add :-6 to the above command which installs wpan tools, but that is optional.



```
alekos@alekos-X555LB: ~/mininet-wifi
File Edit View Search Terminal Help
alekos@alekos-X555LB:~$ git clone git://github.com/intrig-unicamp/mininet-wifi
Cloning into 'mininet-wifi'...
remote: Enumerating objects: 25095, done.
remote: Counting objects: 100% (444/444), done.
remote: Compressing objects: 100% (303/303), done.
remote: Total 25095 (delta 283), reused 267 (delta 141), pack-reused 24651
Receiving objects: 100% (25095/25095), 22.65 MiB | 4.74 MiB/s, done.
Resolving deltas: 100% (17565/17565), done.
alekos@alekos-X555LB:~$ cd mininet-wifi
alekos@alekos-X555LB:~/mininet-wifi$ sudo util/install.sh -Wlnfv
```

**Image 6.8 Mininet-wifi complete installation**

After the process is finished, you can use the “sudo mn --wifi” command to test if the installation was successful.



```
alekos@alekos-X555LB:~/mininet-wifi$ sudo mn --wifi
*** Adding stations:
sta1 sta2
*** Adding access points:
ap1
*** Configuring wifi nodes...
Mac address(es) added into /etc/NetworkManager/conf.d/unmanaged.conf
Restarting network-manager...
*** Creating network
*** Adding controller
*** Adding hosts:

*** Adding switches:

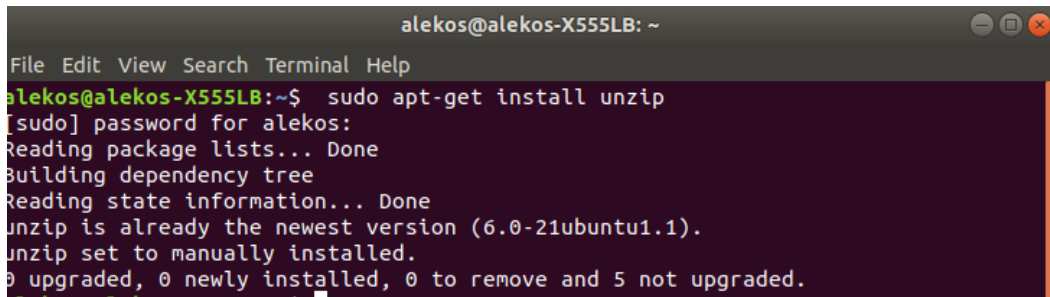
*** Adding links:
(sta1, ap1) (sta2, ap1)
*** Starting controller(s)
c0
*** Starting L2 nodes
ap1 ...
*** Starting CLI:
mininet-wifi>
```

**Image 6.9 Test Mininet-wifi installation**

## 6.4 D-ITG download

Open your terminal(shortcut cntr+alt+T) and type the following commands in order.

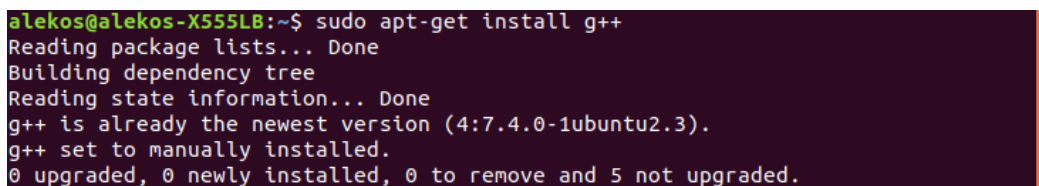
Type “sudo apt-get install unzip”.(in this case it was already installed)



```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo apt-get install unzip  
[sudo] password for alekos:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
unzip is already the newest version (6.0-21ubuntu1.1).  
unzip set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
```

**Image 6.10 Unzip download**

Then install g++ with the command “sudo apt-get install g++”.(in this case it was already installed)



```
alekos@alekos-X555LB:~$ sudo apt-get install g++  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
g++ is already the newest version (4:7.4.0-1ubuntu2.3).  
g++ set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
```

**Image 6.11 G++ download**

After that type “wget http://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip” to get the zip for the source code of D-ITG.



```
alekos@alekos-X555LB:~$ wget http://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip  
--2021-10-29 13:01:09-- http://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip  
Resolving traffic.comics.unina.it (traffic.comics.unina.it)... 143.225.81.79  
Connecting to traffic.comics.unina.it (traffic.comics.unina.it)|143.225.81.79|:80... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 289587 (283K) [application/zip]  
Saving to: 'D-ITG-2.8.1-r1023-src.zip'  
  
D-ITG-2.8.1-r1023-src.zip 100%[=====]  
=====>] 282.80K 564KB/s in 0.5s  
  
2021-10-29 13:01:13 (564 KB/s) - 'D-ITG-2.8.1-r1023-src.zip' saved [289587/289587]
```

**Image 6.12 Get D-ITG zipped source code**

Then unzip the above file with the command “unzip D-ITG-2.8.1-r1023-src.zip”.

```
alekos@alekos-X555LB:~$ unzip D-ITG-2.8.1-r1023-src.zip
Archive: D-ITG-2.8.1-r1023-src.zip
  creating: D-ITG-2.8.1-r1023/
  inflating: D-ITG-2.8.1-r1023/README
  inflating: D-ITG-2.8.1-r1023/CHANGELOG
  extracting: D-ITG-2.8.1-r1023/REVISION
  creating: D-ITG-2.8.1-r1023/bin/
```

**Image 6.13 Unzip source file**

Lastly, type “cd D-ITG-2.8.1-r1023/src” and then use the command “make”.

```
alekos@alekos-X555LB:~$ cd D-ITG-2.8.1-r1023/src
alekos@alekos-X555LB:~/D-ITG-2.8.1-r1023/src$ make

-----
Checking dependencies
-----
All dependencies satisfied.

-----
Current configuration
-----
Enabled features*: bursty multiport
Version: 2.8.1
Revision: 1023
Compiler: g++
OS: Linux
OS Flags: -DUNIX
```

**Image 6.14 Make**

If the process was successful the following will appear in the terminal.

```
D-ITG executables created in /home/alekos/D-ITG-2.8.1-r1023/bin
alekos@alekos-X555LB:~/D-ITG-2.8.1-r1023/src$
```

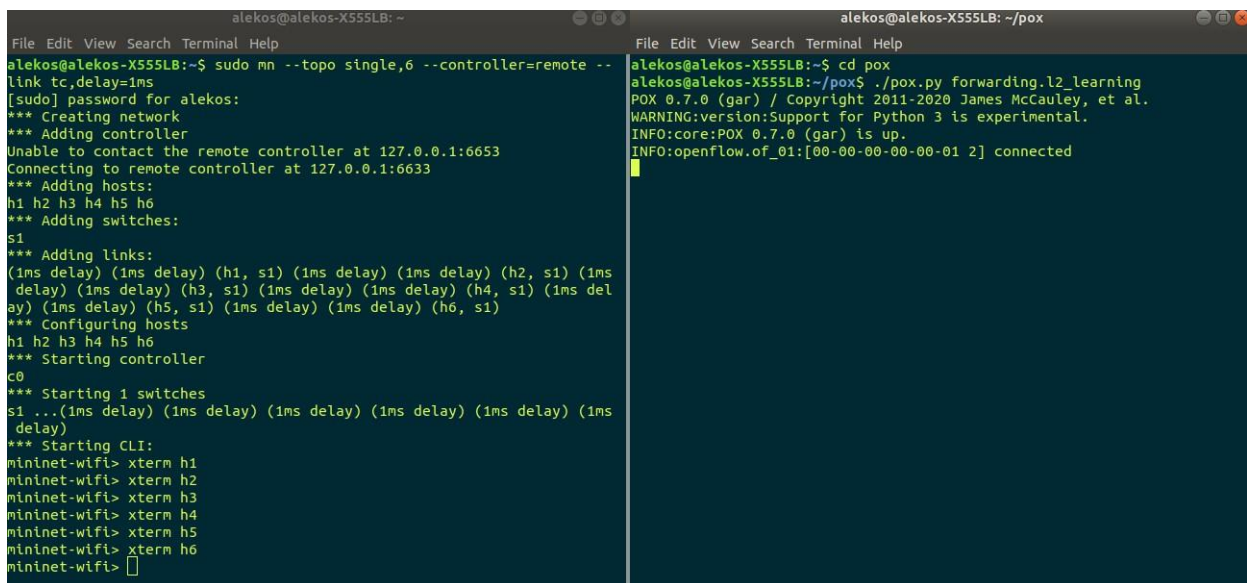
**Image 6.15 Successful installation message**

## 7. Experiments in Traffic Generation

### 7.1 First Experiment: 6-18 Host Topology

In our first experiment we check the operation, as well as, quality metrics' difference between a simple network with 6 hosts and one with 18 hosts.

Firstly, we activate POX and then we create a simple network topology, as shown in chapter 5. The forwarding.l2\_learning component is used for pox, whereas for the network, with the help of Mininet-wifi, we establish a 6 host, single switch topology, with a remote controller(pox) and we also added an 1ms delay in the links in order for the results to be more real-life accurate.



```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo mn --topo single,6 --controller=remote --  
link tc,delay=1ms  
[sudo] password for alekos:  
*** Creating network  
*** Adding controller  
Unable to contact the remote controller at 127.0.0.1:6653  
Connecting to remote controller at 127.0.0.1:6633  
*** Adding hosts:  
h1 h2 h3 h4 h5 h6  
*** Adding switches:  
s1  
*** Adding links:  
(1ms delay) (1ms delay) (h1, s1) (1ms delay) (1ms delay) (h2, s1) (1ms  
delay) (1ms delay) (h3, s1) (1ms delay) (1ms delay) (h4, s1) (1ms del  
ay) (1ms delay) (h5, s1) (1ms delay) (1ms delay) (h6, s1)  
*** Configuring hosts  
h1 h2 h3 h4 h5 h6  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ... (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms  
delay)  
*** Starting CLI:  
mininet-wifi> xterm h1  
mininet-wifi> xterm h2  
mininet-wifi> xterm h3  
mininet-wifi> xterm h4  
mininet-wifi> xterm h5  
mininet-wifi> xterm h6  
mininet-wifi>   
alekos@alekos-X555LB: ~/pox  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~/pox$ ./pox.py forwarding.l2_learning  
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.  
WARNING:version:Support for Python 3 is experimental.  
INFO:core:POX 0.7.0 (gar) is up.  
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Image 7.1 Six Host topology and Pox activation(1st exp./pt1)

After the creation of the network was complete, we opened xterms for all the hosts and used of D-ITG to emulate traffic.

The image shows two terminal windows side-by-side, labeled "Node: h1" and "Node: h6".

**Node: h1** shows the execution of the following commands:

```
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend -T UDP -a 10.0.0.6 -c 600
-C 850 -t 20000 -l sender.log -x receiver.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Started sending packets of flow ID: 1
Finished sending packets of flow ID: 1

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
```

The output for Node h1 shows flow statistics:

```
Flow number: 1
From 10.0.0.1:36227
To 10.0.0.6:8999

-----
Total time = 19.999729 s
Total packets = 15707
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 9424200
Average bitrate = 3769.731080 Kbit/s
Average packet rate = 785.360642 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

***** TOTAL RESULTS *****
Number of flows = 1
Total time = 19.999729 s
Total packets = 15707
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 9424200
Average bitrate = 3769.731080 Kbit/s
Average packet rate = 785.360642 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
```

**Node: h6** shows the execution of the following commands:

```
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 8999
Finish on UDP port : 8999
^CFinish with CTRL-C!

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec receiver.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
```

The output for Node h6 shows flow statistics:

```
Flow number: 1
From 10.0.0.1:36227
To 10.0.0.6:8999

-----
Total time = 19.998096 s
Total packets = 15707
Minimum delay = 0.002034 s
Maximum delay = 0.004050 s
Average delay = 0.002162 s
Average jitter = 0.000023 s
Delay standard deviation = 0.000058 s
Bytes received = 9424200
Average bitrate = 3770.038908 Kbit/s
Average packet rate = 785.424772 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

***** TOTAL RESULTS *****
Number of flows = 1
Total time = 19.998096 s
Total packets = 15707
Minimum delay = 0.002034 s
Maximum delay = 0.004050 s
Average delay = 0.002162 s
Average jitter = 0.000023 s
Delay standard deviation = 0.000058 s
Bytes received = 9424200
Average bitrate = 3770.038908 Kbit/s
Average packet rate = 785.424772 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
```

At the bottom of the Node h6 terminal, the command `./ITGDec receiver.log -c 500 f1_6.dat` is shown, which creates a .dat file with the average metrics of the flow.

**Image 7.2 D-ITG Simple traffic generation(1st exp./pt1)**

Firstly, the receiver host was created with the command: `./ITGRecv`.

(depending on the file config one has done you might need to access the correct folder before typing commands e.g `cd D-ITG-2.8.1-r1023/bin`)

Then with the command:

`./ITGSend -T UDP -a 10.0.0.6 -c 600 -C 850 -t 20000 -l sender.log -x receiver.log`

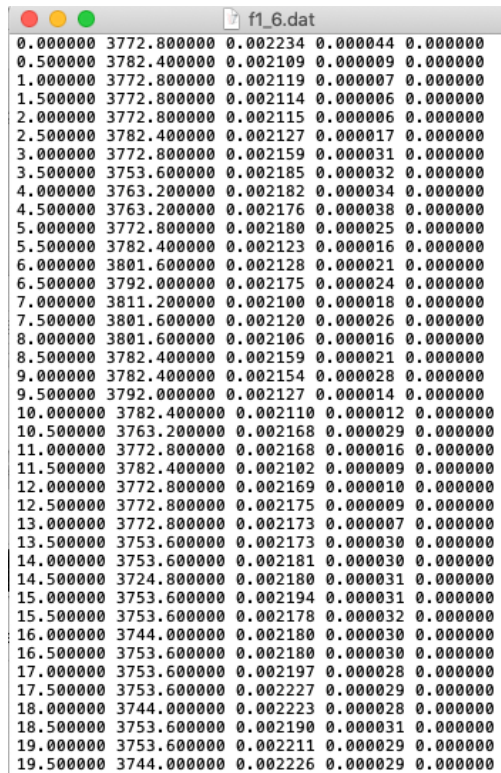
We created a single UDP type flow, to the host with IP address 10.0.0.6, with a packet size of 600 bytes, with a rate of 850pkts/s. The duration of the generation is 20000ms and we also set here a sender logfile, with the name sender.log and a respective one for the receiver.

After the conclusion of the generation, we used the Dec component, to decode the log files we created and see all the stats/metrics, with the command `./ITGDec sender.log` and `./ITGDec receiver.log` respectively.

Lastly, the command `./ITGDec receiver.log -c 500 f1_6.dat` is used to create a .dat file with the average metrics of the flow. It collects results every 500ms.

Each line of the output file respectively contains the following fields: - "**Time**", "**Bitrate**", "**Delay**", "**Jitter**", "**Packet Loss**".

The .dat file is shown below, as well as, the diagrams of each quality metric.



```

0.000000 3772.800000 0.002234 0.000044 0.000000
0.500000 3782.400000 0.002109 0.000009 0.000000
1.000000 3772.800000 0.002119 0.000007 0.000000
1.500000 3772.800000 0.002114 0.000006 0.000000
2.000000 3772.800000 0.002115 0.000006 0.000000
2.500000 3782.400000 0.002127 0.000017 0.000000
3.000000 3772.800000 0.002159 0.000031 0.000000
3.500000 3753.600000 0.002185 0.000032 0.000000
4.000000 3763.200000 0.002182 0.000034 0.000000
4.500000 3763.200000 0.002176 0.000038 0.000000
5.000000 3772.800000 0.002180 0.000025 0.000000
5.500000 3782.400000 0.002123 0.000016 0.000000
6.000000 3801.600000 0.002128 0.000021 0.000000
6.500000 3792.000000 0.002175 0.000024 0.000000
7.000000 3811.200000 0.002100 0.000018 0.000000
7.500000 3801.600000 0.002120 0.000026 0.000000
8.000000 3801.600000 0.002106 0.000016 0.000000
8.500000 3782.400000 0.002159 0.000021 0.000000
9.000000 3782.400000 0.002154 0.000028 0.000000
9.500000 3792.000000 0.002127 0.000014 0.000000
10.000000 3782.400000 0.002110 0.000012 0.000000
10.500000 3763.200000 0.002168 0.000029 0.000000
11.000000 3772.800000 0.002168 0.000016 0.000000
11.500000 3782.400000 0.002102 0.000009 0.000000
12.000000 3772.800000 0.002169 0.000010 0.000000
12.500000 3772.800000 0.002175 0.000009 0.000000
13.000000 3772.800000 0.002173 0.000007 0.000000
13.500000 3753.600000 0.002173 0.000030 0.000000
14.000000 3753.600000 0.002181 0.000030 0.000000
14.500000 3724.800000 0.002180 0.000031 0.000000
15.000000 3753.600000 0.002194 0.000031 0.000000
15.500000 3753.600000 0.002178 0.000032 0.000000
16.000000 3744.000000 0.002180 0.000030 0.000000
16.500000 3753.600000 0.002180 0.000030 0.000000
17.000000 3753.600000 0.002197 0.000028 0.000000
17.500000 3753.600000 0.002227 0.000029 0.000000
18.000000 3744.000000 0.002223 0.000028 0.000000
18.500000 3753.600000 0.002190 0.000031 0.000000
19.000000 3753.600000 0.002211 0.000029 0.000000
19.500000 3744.000000 0.002226 0.000029 0.000000

```

**Image 7.3 dat File(1st exp./pt1)**

The diagrams were made in Matlab, with the use of the following script, each time changing the value of the integral in the y parameter to “browse through” the metrics’ columns.

```

load f1_6.dat
x=f1_6(:,1);
y=f1_6(:,2);

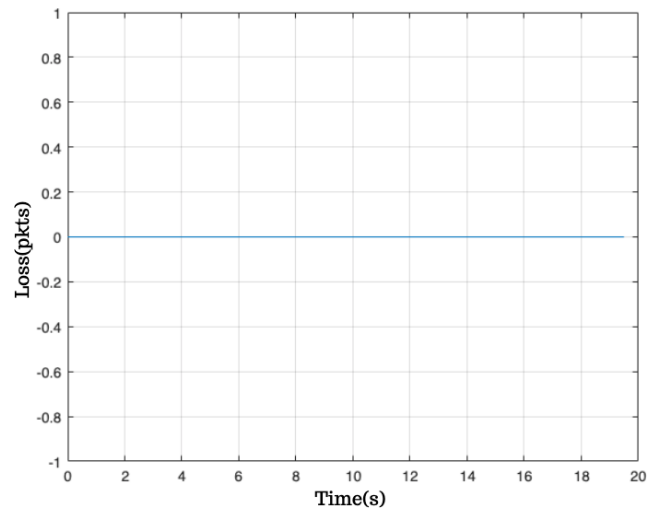
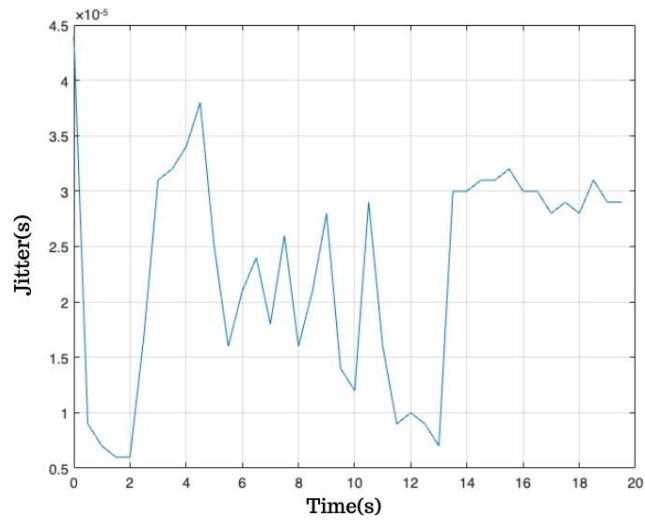
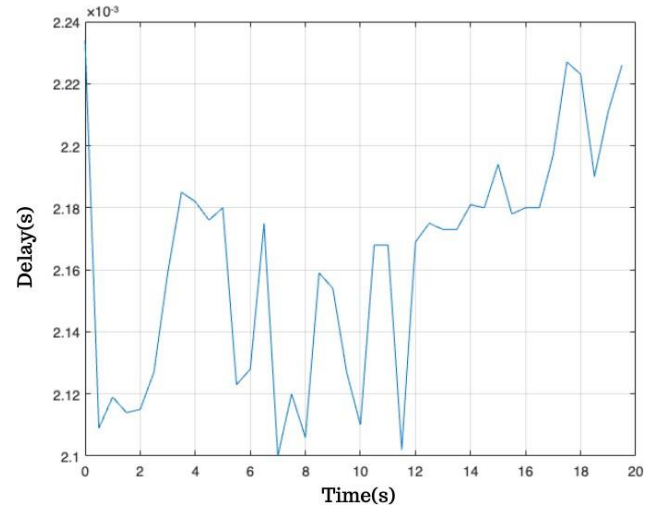
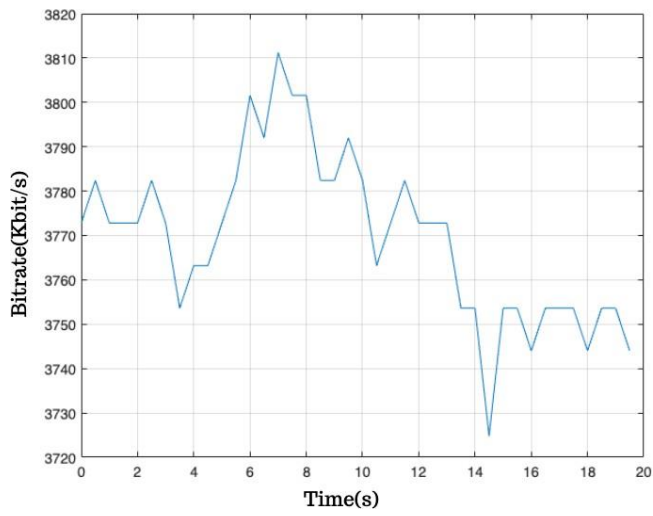
plot(x,y);
grid
figure(1);

```

**Image 7.4 Matlab script(1st exp./pt.1)**



## DIAGRAMS



**Image 7.5 Metrics' diagrams(1st exp./pt1)**

The same process was repeated for the remaining hosts, bearing the change of the IP addresses and the packet size and rate.

For the second part of this experiment, we replicated what was implemented above, with the difference being the number of hosts in the network, this time being 18.

```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo mn --topo single,18 --controller=remote --link tc,delay=1ms  
[sudo] password for alekos:  
*** Creating network  
*** Adding controller  
Unable to contact the remote controller at 127.0.0.1:6653  
Connecting to remote controller at 127.0.0.1:6633  
*** Adding hosts:  
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18  
*** Adding switches:  
s1  
*** Adding links:  
(1ms delay) (1ms delay) (h1, s1) (1ms delay) (1ms delay) (h2, s1) (1ms delay) (1ms delay)  
(h3, s1) (1ms delay) (1ms delay) (h4, s1) (1ms delay) (1ms delay) (h5, s1) (1ms delay) (1ms  
delay) (h6, s1) (1ms delay) (1ms delay) (h7, s1) (1ms delay) (1ms delay) (h8, s1) (1ms d  
elay) (1ms delay) (h9, s1) (1ms delay) (1ms delay) (h10, s1) (1ms delay) (1ms delay) (h11,  
s1) (1ms delay) (1ms delay) (h12, s1) (1ms delay) (1ms delay) (h13, s1) (1ms delay) (1ms  
delay) (h14, s1) (1ms delay) (1ms delay) (h15, s1) (1ms delay) (1ms delay) (h16, s1) (1ms  
delay) (1ms delay) (h17, s1) (1ms delay) (1ms delay) (h18, s1)  
*** Configuring hosts  
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ... (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay)  
(1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms d  
elay) (1ms delay) (1ms delay) (1ms delay)  
*** Starting CLI:  
mininet-wifi> xterm h1  
mininet-wifi> xterm h2  
mininet-wifi> xterm h3  
mininet-wifi> xterm h4  
mininet-wifi> xterm h5  
mininet-wifi> xterm h6  
mininet-wifi> xterm h7  
mininet-wifi> xterm h8  
mininet-wifi> xterm h9  
mininet-wifi> xterm h10  
mininet-wifi> xterm h11  
mininet-wifi> xterm h12  
mininet-wifi> xterm h13  
mininet-wifi> xterm h14  
mininet-wifi> xterm h15  
mininet-wifi> xterm h16  
mininet-wifi> xterm h17  
mininet-wifi> xterm h18
```

**Image 7.6 Eighteen Host topology and Pox activation(1st exp./pt2)**

This time as well, we generated traffic in “duos”, between every host of the network retrieved the results.

```

root@alekos-X559LB:~# cd /D-ITG-2.8.1-r1023/bin
root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend -t UDP -a 10.0.0.18 -c 50
0 -C 850 -t 20000 -l sender_log -x receiver.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Started sending packets of flow ID: 1
Finished sending packets of flow ID: 1

root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
/-----/
Flow number: 1
From 10.0.0.1:40509
To 10.0.0.18:8999

-----
Total time = 19.999875 s
Total packets = 15640
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 7820000
Average bitrate = 3128.019550 Kbit/s
Average packet rate = 782.004888 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt

-----
***** TOTAL RESULTS *****
-----
Number of flows = 1
Total time = 19.999875 s
Total packets = 15640
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 7820000
Average bitrate = 3128.019550 Kbit/s
Average packet rate = 782.004888 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0

-----
root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin#

root@alekos-X559LB:~# cd /D-ITG-2.8.1-r1023/bin
root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 8999
Finish on UDP port : 8999
^CFinish with Ctrl-C

root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec receiver.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
/-----/
Flow number: 1
From 10.0.0.1:40509
To 10.0.0.18:8999

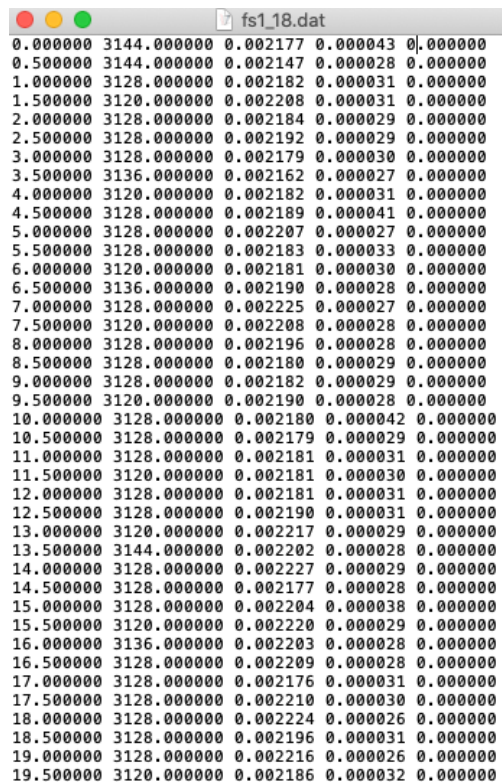
-----
Total time = 19.998308 s
Total packets = 15640
Minimum delay = 0.002041 s
Maximum delay = 0.003910 s
Average delay = 0.002133 s
Average jitter = 0.000030 s
Delay standard deviation = 0.000050 s
Bytes received = 7820000
Average bitrate = 3128.264651 Kbit/s
Average packet rate = 782.066163 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt

-----
***** TOTAL RESULTS *****
-----
Number of flows = 1
Total time = 19.998308 s
Total packets = 15640
Minimum delay = 0.002041 s
Maximum delay = 0.003910 s
Average delay = 0.002133 s
Average jitter = 0.000030 s
Delay standard deviation = 0.000050 s
Bytes received = 7820000
Average bitrate = 3128.264651 Kbit/s
Average packet rate = 782.066163 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0

-----
root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec receiver.log -c 50 fsi-18.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X559LB:~/D-ITG-2.8.1-r1023/bin#
```

**Image 7.7 D-ITG Simple traffic generation(1st exp./pt2)**

The .dat file is shown below, as well as, the diagrams of each quality metric.



```
fs1_18.dat
0.000000 3144.000000 0.002177 0.000043 0.000000
0.500000 3144.000000 0.002147 0.000028 0.000000
1.000000 3128.000000 0.002182 0.000031 0.000000
1.500000 3120.000000 0.002208 0.000031 0.000000
2.000000 3128.000000 0.002184 0.000029 0.000000
2.500000 3128.000000 0.002192 0.000029 0.000000
3.000000 3128.000000 0.002179 0.000030 0.000000
3.500000 3136.000000 0.002162 0.000027 0.000000
4.000000 3120.000000 0.002182 0.000031 0.000000
4.500000 3128.000000 0.002189 0.000041 0.000000
5.000000 3128.000000 0.002207 0.000027 0.000000
5.500000 3128.000000 0.002183 0.000033 0.000000
6.000000 3120.000000 0.002181 0.000030 0.000000
6.500000 3136.000000 0.002190 0.000028 0.000000
7.000000 3128.000000 0.002225 0.000027 0.000000
7.500000 3120.000000 0.002208 0.000028 0.000000
8.000000 3128.000000 0.002196 0.000028 0.000000
8.500000 3128.000000 0.002180 0.000029 0.000000
9.000000 3128.000000 0.002182 0.000029 0.000000
9.500000 3120.000000 0.002190 0.000028 0.000000
10.000000 3128.000000 0.002180 0.000042 0.000000
10.500000 3128.000000 0.002179 0.000029 0.000000
11.000000 3128.000000 0.002181 0.000031 0.000000
11.500000 3120.000000 0.002181 0.000030 0.000000
12.000000 3128.000000 0.002181 0.000031 0.000000
12.500000 3128.000000 0.002190 0.000031 0.000000
13.000000 3120.000000 0.002217 0.000029 0.000000
13.500000 3144.000000 0.002202 0.000028 0.000000
14.000000 3128.000000 0.002227 0.000029 0.000000
14.500000 3128.000000 0.002177 0.000028 0.000000
15.000000 3128.000000 0.002204 0.000038 0.000000
15.500000 3120.000000 0.002220 0.000029 0.000000
16.000000 3136.000000 0.002203 0.000028 0.000000
16.500000 3128.000000 0.002209 0.000028 0.000000
17.000000 3128.000000 0.002176 0.000031 0.000000
17.500000 3128.000000 0.002210 0.000030 0.000000
18.000000 3128.000000 0.002224 0.000026 0.000000
18.500000 3128.000000 0.002196 0.000031 0.000000
19.000000 3128.000000 0.002216 0.000026 0.000000
19.500000 3120.000000 0.002186 0.000032 0.000000
```

**Image 7.8 dat File(1st exp./pt.2)**

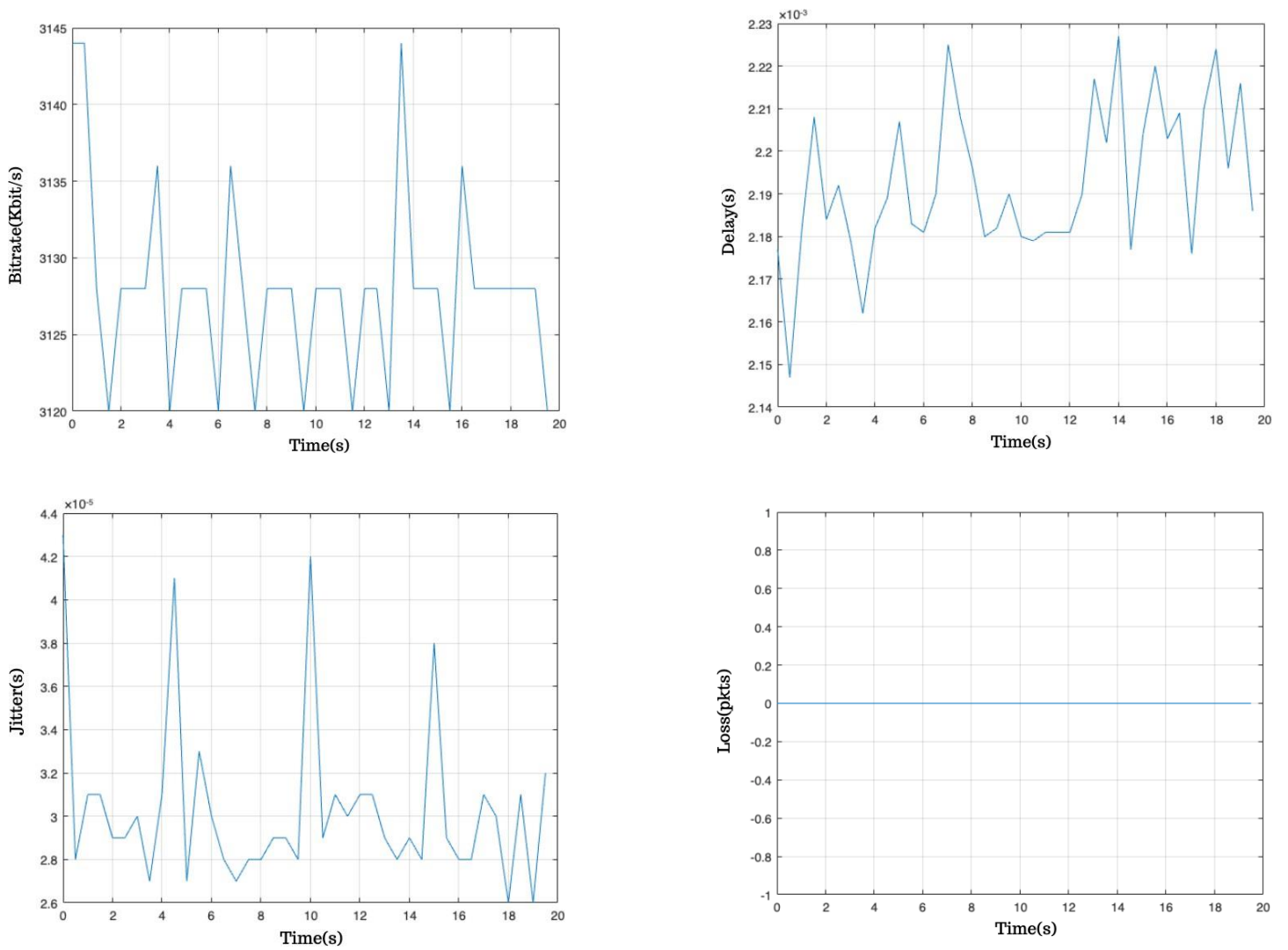
The Matlab script:

```
load fs1_18.dat
x=fs1_18(:,1);
y=fs1_18(:,5);

plot(x,y);
grid
figure(1);
```

**Image 7.9 Matlab script(1st exp./pt.2)**

## DIAGRAMS



**Image 7.10 Metrics' diagrams(1st exp./pt.2)**

The same process was repeated for the remaining hosts, bearing the change of the IP addresses and the packet size and rate.

We measured the medium of each average metric for each of the networks and compared them. The loss is not measured as it was set at 0, at the creation of the topology.

	6host topo	18 host topo
<b>Delay</b>	0,002180666666667s	0,002203333333333s
<b>Jitter</b>	0,000023666666667s	0,000027s
<b>Bitrate</b>	4198,396648333333333kbit/s	3356,599432111111kbit/s

**Table 7.1 Average Metrics**

As we expected the metrics in the network with the fewer host operates with a higher quality.

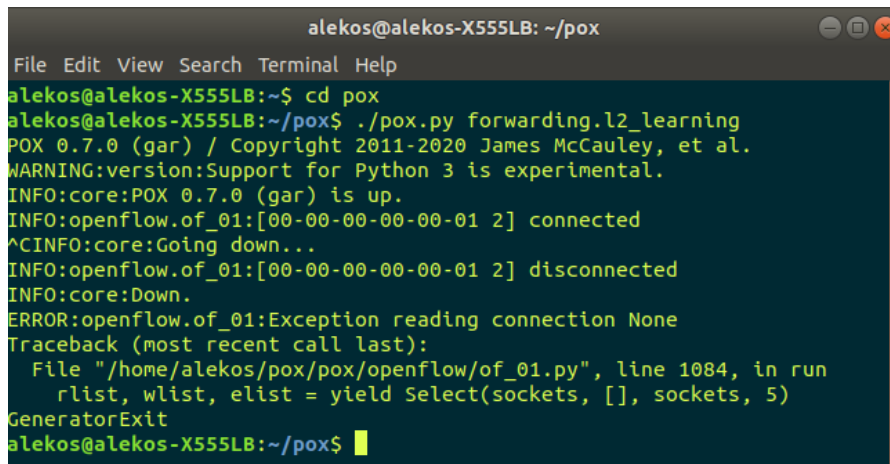
Note:

It is highly important to exit Mininet(-wifi) and then execute the command `sudo mn -c`, when one is done with the network, in order for there to be no implications.

```
mininet-wifi> exit
*** Stopping 1 controllers
c0
*** Stopping 6 terms
*** Stopping 6 links
.....
*** Stopping 1 switches
s1
*** Stopping 6 hosts
h1 h2 h3 h4 h5 h6
*** Done
completed in 1458.471 seconds
alekos@galekos-X555LB:~$ sudo mn -c
[sudo] password for alekos:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_.:alnum:])+eth[[:digit:]]+'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```

**Image 7.11 Mininet exit and cleanup**

Furthermore, you should always turn POX off, with the shortcut: command+C, or else the connection is still going to run in the background .

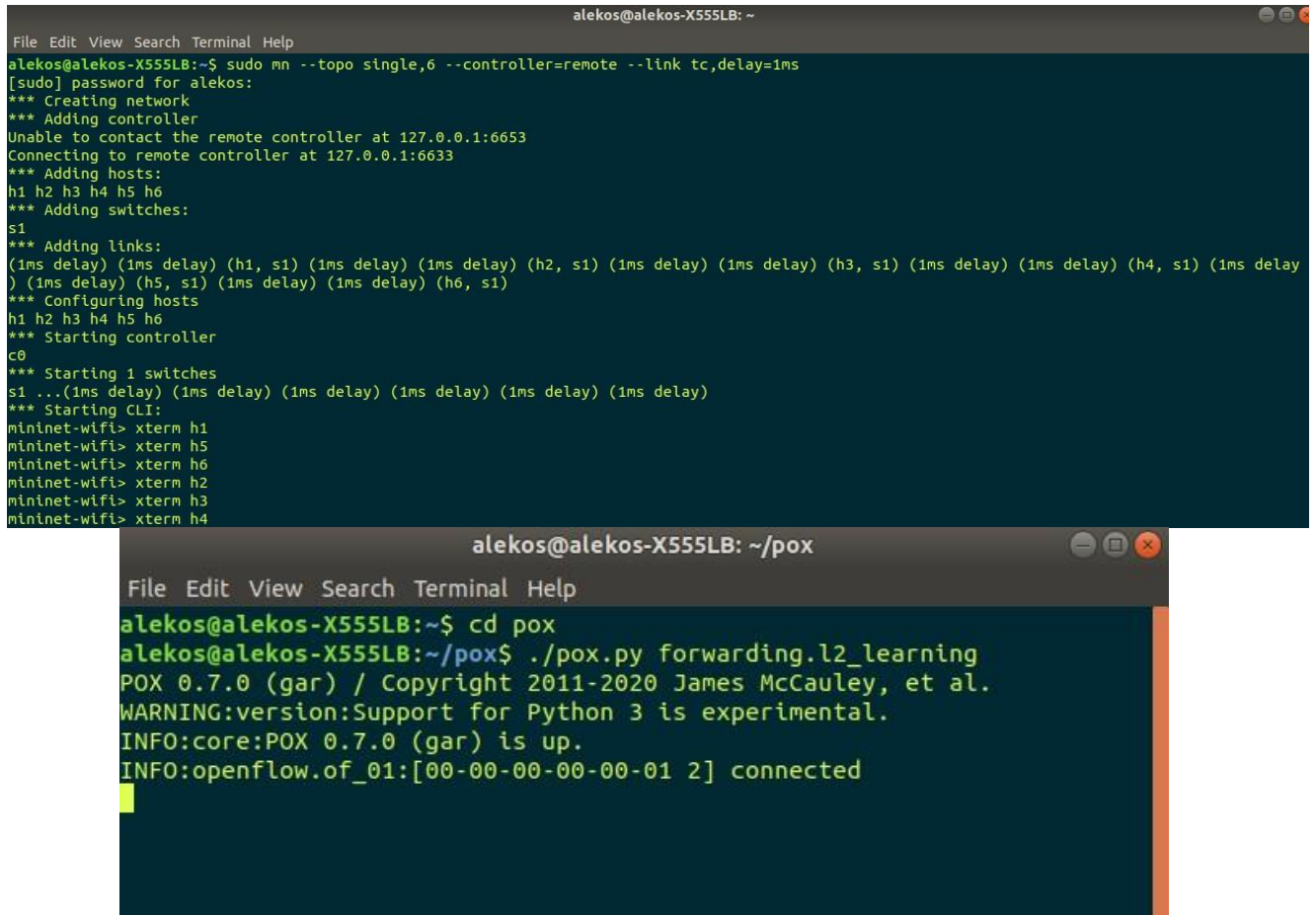
A terminal window titled 'alekos@alekos-X555LB: ~/pox' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the execution of './pox.py forwarding.l2\_learning'. It displays version information for POX 0.7.0 (gar) and a warning about Python 3 support. It then shows a connection being established and subsequently disconnected, followed by a shutdown sequence. An error message is shown: 'Exception reading connection None'. A traceback follows, pointing to line 1084 in 'of\_01.py'. The process ends with 'GeneratorExit' and the prompt returns to 'alekos@alekos-X555LB: ~/pox\$'.

```
alekos@alekos-X555LB:~/pox$ cd pox
alekos@alekos-X555LB:~/pox$ ./pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 2] disconnected
INFO:core:Down.
ERROR:openflow.of_01:Exception reading connection None
Traceback (most recent call last):
  File "/home/alekos/pox/pox/openflow/of_01.py", line 1084, in run
    rlist, wlist, elist = yield Select(sockets, [], sockets, 5)
GeneratorExit
alekos@alekos-X555LB:~/pox$
```

**Image 7.12 Turn POX down**

## 7.2 Second Experiment: Multiple Flows Networks

In our second experiment we created networks, in which the hosts communicate through one or more traffic flows. In the first part, we opened POX ,then we created a 6 host topology just like the one on the first experiment and opened the xterms for the hosts.



The image shows two terminal windows. The top window is titled 'alekos@alekos-X555LB: ~' and shows the execution of 'sudo mn --topo single,6 --controller=remote --link tc,delay=1ms'. It displays the process of creating a network with 6 hosts (h1-h6) and 1 switch (s1), adding links with delays, and starting the controller and CLI. The bottom window is titled 'alekos@alekos-X555LB: ~/pox' and shows the execution of 'cd pox' and './pox.py forwarding.l2\_learning'. It displays the POX version (0.7.0), copyright information, and connection status for OpenFlow.

```
alekos@alekos-X555LB:~$ sudo mn --topo single,6 --controller=remote --link tc,delay=1ms
[sudo] password for alekos:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(1ms delay) (1ms delay) (h1, s1) (1ms delay) (1ms delay) (h2, s1) (1ms delay) (1ms delay) (h3, s1) (1ms delay) (1ms delay) (h4, s1) (1ms delay)
(1ms delay) (h5, s1) (1ms delay) (1ms delay) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ... (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay)
*** Starting CLI:
mininet-wifi> xterm h1
mininet-wifi> xterm h5
mininet-wifi> xterm h6
mininet-wifi> xterm h2
mininet-wifi> xterm h3
mininet-wifi> xterm h4

alekos@alekos-X555LB:~/pox
File Edit View Search Terminal Help
alekos@alekos-X555LB:~$ cd pox
alekos@alekos-X555LB:~/pox$ ./pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Image 7.13 Six Host topology and Pox activation(2nd exp./pt.1)

This time we created to Receiver instances in hosts 5 and 6 with the `./ITGRecv -l recv1_log_file` and `./ITGRecv -l recv2_log_file` commands respectively, which also create separate log files for each receiver.

Then we executed the command `./ITGSend script_file -l sender.log`, which creates traffic according to the specified script file. In this case the script used was:

```
-a 10.0.0.6 -rp 10001 VoIP -x G.711.2 -h RTP -VAD
-a 10.0.0.6 -rp 10002 Telnet
-a 10.0.0.5 -rp 10003 DNS
```

Image 7.14 Custom-Flow script (2nd exp./pt.1.1)

The above script creates two flows between the sender host h1 and the receiver h6 & one flow with the receiver h5. The address Its are specified along the resource ports. Each flow emulates a different type of connection, with VoIp, Telnet and DNS being the ones in this experiment. Lastly, as on the first experiment the .dat files with the metrics are shown.



```
"Node: h1"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend script_file -l sender.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Voice Codec: G.711
Framesize: 80,00
Samples: 2
Packets per sec.: 50
VAD: Si
Started sending packets of flow ID: 2
Started sending packets of flow ID: 1
Started sending packets of flow ID: 3
Finished sending packets of flow ID: 2
Finished sending packets of flow ID: 3
Finished sending packets of flow ID: 1

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 1
From 10,0,0,1:33852
To 10,0,0,6:10001
-----
Total time = 9,979823 s
Total packets = 496
Minimum delay = 0,000000 s
Maximum delay = 0,000000 s
Average delay = 0,000000 s
Average jitter = 0,000000 s
Delay standard deviation = 0,000000 s
Bytes received = 57536
Average bitrate = 46,121850 Kbit/s
Average packet rate = 49,700280 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0,000000 pkt
-----
Flow number: 2
From 10,0,0,1:48400
To 10,0,0,6:10002
-----
Total time = 9,998541 s
Total packets = 543
Minimum delay = 0,000000 s
Maximum delay = 0,000000 s
Average delay = 0,000000 s
Average jitter = 0,000000 s
Delay standard deviation = 0,000000 s
Bytes received = 1164
Average bitrate = 0,331327 Kbit/s
Average packet rate = 54,307380 pkt/s
-----
Flow number: 3
From 10,0,0,1:44022
To 10,0,0,5:10003
-----
Total time = 8,936300 s
Total packets = 6
Minimum delay = 0,000000 s
Maximum delay = 0,000000 s
Average delay = 0,000000 s
Average jitter = 0,000000 s
Delay standard deviation = 0,000000 s
Bytes received = 1225
Average bitrate = 1,096551 Kbit/s
Average packet rate = 0,671419 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0,000000 pkt
-----
***** TOTAL RESULTS *****
Number of flows = 3
Total time = 10,013296 s
Total packets = 1045
Minimum delay = 0,000000 s
Maximum delay = 0,000000 s
Average delay = 0,000000 s
Average jitter = 0,000000 s
Delay standard deviation = 0,000000 s
Bytes received = 59325
Average bitrate = 47,875344 Kbit/s
Average packet rate = 104,361241 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0 pkt
Error lines = 0

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

```
"Node: h5"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv1_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 10003
Finish on UDP port : 10003
^CFinish with CTRL-C!
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv1_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 3
From 10,0,0,1:44022
To 10,0,0,5:10003
-----
Total time = 8,934832 s
Total packets = 6
Minimum delay = 0,002116 s
Maximum delay = 0,003812 s
Average delay = 0,002562 s
Average jitter = 0,000385 s
Delay standard deviation = 0,000575 s
Bytes received = 1225
Average bitrate = 1,096831 Kbit/s
Average packet rate = 0,671523 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0,000000 pkt
-----
***** TOTAL RESULTS *****
Number of flows = 1
Total time = 8,934832 s
Total packets = 6
Minimum delay = 0,002116 s
Maximum delay = 0,003812 s
Average delay = 0,002562 s
Average jitter = 0,000385 s
Delay standard deviation = 0,000575 s
Bytes received = 1225
Average bitrate = 1,096831 Kbit/s
Average packet rate = 0,671523 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0 pkt
Error lines = 0

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv1_log_file -c 500 sl_5
.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#

"Node: h6"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv2_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on TCP port : 10002
Listening on UDP port : 10001
Finish on TCP port : 10002
Finish on UDP port : 10001
^CFinish with CTRL-C!
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv2_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 1
From 10,0,0,1:33852
To 10,0,0,6:10001
-----
Total time = 9,978832 s
Total packets = 496
Minimum delay = 0,002070 s
Maximum delay = 0,003240 s
Average delay = 0,002265 s
Average jitter = 0,000385 s
Delay standard deviation = 0,000575 s
Bytes received = 57536
Average bitrate = 46,126440 Kbit/s
Average packet rate = 49,705216 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0,000000 pkt
-----
Flow number: 2
From 10,0,0,1:48400
To 10,0,0,6:10002
-----
Total time = 9,999584 s
Total packets = 543
Minimum delay = 0,002074 s
Maximum delay = 0,006703 s
Average delay = 0,003824 s
Average jitter = 0,001719 s
Delay standard deviation = 0,001390 s
Bytes received = 1164
Average bitrate = 0,331238 Kbit/s
Average packet rate = 54,302253 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0,000000 pkt
-----
***** TOTAL RESULTS *****
Number of flows = 2
Total time = 10,013367 s
Total packets = 1039
Minimum delay = 0,002070 s
Maximum delay = 0,006703 s
Average delay = 0,003809 s
Average jitter = 0,000823 s
Delay standard deviation = 0,001273 s
Bytes received = 58700
Average bitrate = 46,897312 Kbit/s
Average packet rate = 103,761302 pkt/s
Packets dropped = 0 (0,00 %)
Average loss-burst size = 0 pkt
Error lines = 0

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv2_log_file -c 500 sl_6
.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

Image 7.15 D-ITG Multi-flow traffic generation(2nd exp./pt.1.1)



The .dat files are shown below, as well as, the diagrams of each quality metric.

s1_5.dat				
0.000000	3.056000	0.003812	0.000000	0.000000
0.500000	0.000000	0.000000	0.000000	0.000000
1.000000	0.000000	0.000000	0.000000	0.000000
1.500000	2.096000	0.002558	0.001254	0.000000
2.000000	0.000000	0.000000	0.000000	0.000000
2.500000	0.000000	0.000000	0.000000	0.000000
3.000000	0.000000	0.000000	0.000000	0.000000
3.500000	3.120000	0.002239	0.000319	0.000000
4.000000	0.000000	0.000000	0.000000	0.000000
4.500000	0.000000	0.000000	0.000000	0.000000
5.000000	4.768000	0.002116	0.000123	0.000000
5.500000	0.000000	0.000000	0.000000	0.000000
6.000000	0.000000	0.000000	0.000000	0.000000
6.500000	0.000000	0.000000	0.000000	0.000000
7.000000	4.800000	0.002302	0.000186	0.000000
7.500000	0.000000	0.000000	0.000000	0.000000
8.000000	0.000000	0.000000	0.000000	0.000000
8.500000	1.760000	0.002344	0.000042	0.000000

s1_6.dat				
0.000000	45.312000	0.006112	0.001676	0.000000
0.500000	44.544000	0.002315	0.000017	0.000000
1.000000	46.400000	0.002315	0.000016	0.000000
1.500000	46.400000	0.002315	0.000018	0.000000
2.000000	46.400000	0.002316	0.000014	0.000000
2.500000	44.544000	0.002314	0.000012	0.000000
3.000000	46.400000	0.002225	0.000043	0.000000
3.500000	47.952000	0.006016	0.001726	0.000000
4.000000	48.576000	0.006018	0.001859	0.000000
4.500000	46.992000	0.006311	0.002122	0.000000
5.000000	46.400000	0.002187	0.000057	0.000000
5.500000	44.544000	0.002233	0.000061	0.000000
6.000000	47.536000	0.006092	0.001633	0.000000
6.500000	48.912000	0.005994	0.001659	0.000000
7.000000	47.504000	0.005683	0.001501	0.000000
7.500000	48.736000	0.006371	0.002088	0.000000
8.000000	46.096000	0.006105	0.001755	0.000000
8.500000	48.112000	0.006147	0.001962	0.000000
9.000000	47.984000	0.006060	0.001501	0.000000
9.500000	48.000000	0.006033	0.001474	0.000000
10.000000	1.856000	0.002249	0.000050	0.000000

Image 7.16 dat Files(2nd exp./pt.1.1)

The first Matlab script :

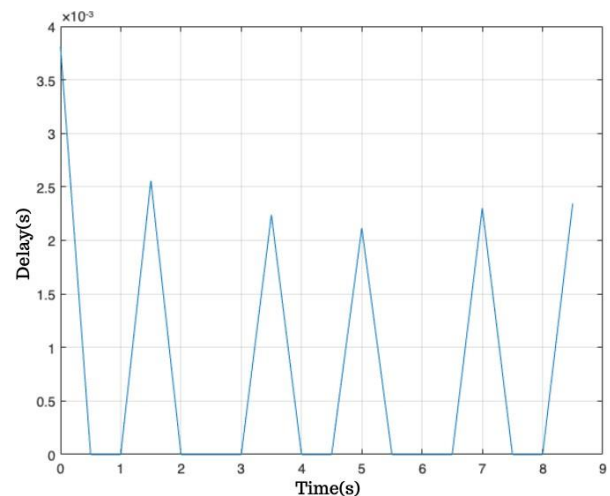
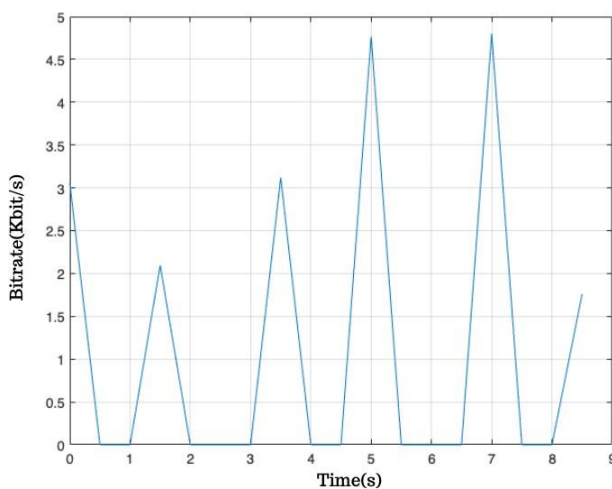
```
load s1_5.dat
x=s1_5(:,1);
y=s1_5(:,5);

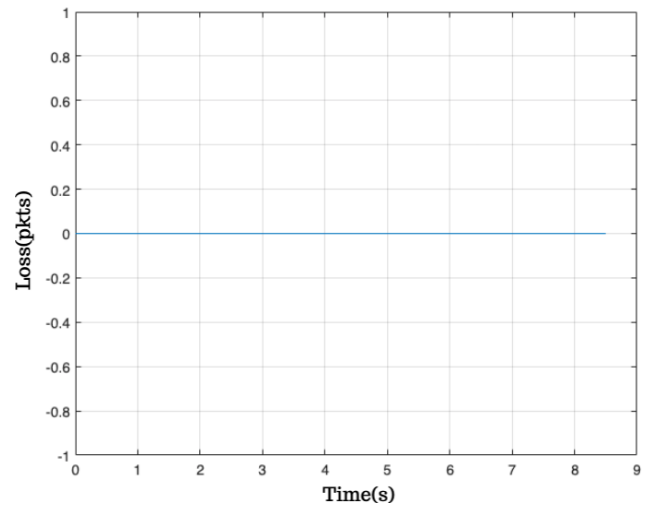
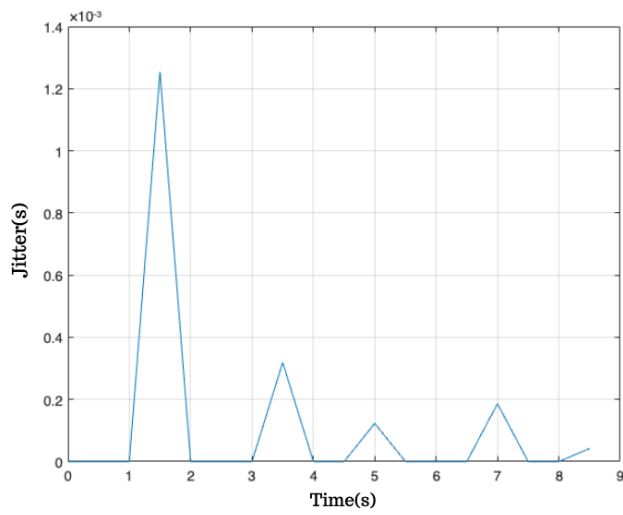
plot(x,y);
grid
figure(1);
```

Image 7.17 Matlab script(2nd exp./pt.1.1.1)

## DIAGRAMS

H1 -> H5





**Image 7.18 Metrics' diagrams(2nd exp./pt.1.1.1)**

The second Matlab script :

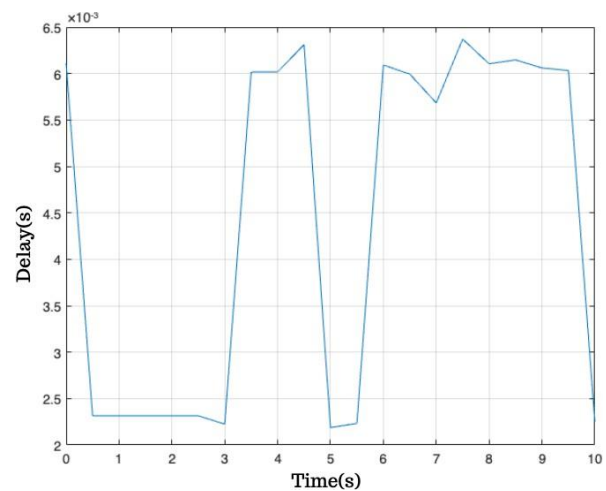
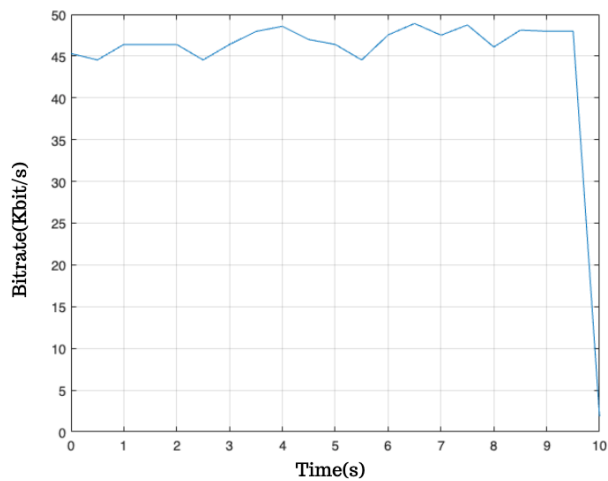
```
load s1_6.dat
x=s1_6(:,1);
y=s1_6(:,2);

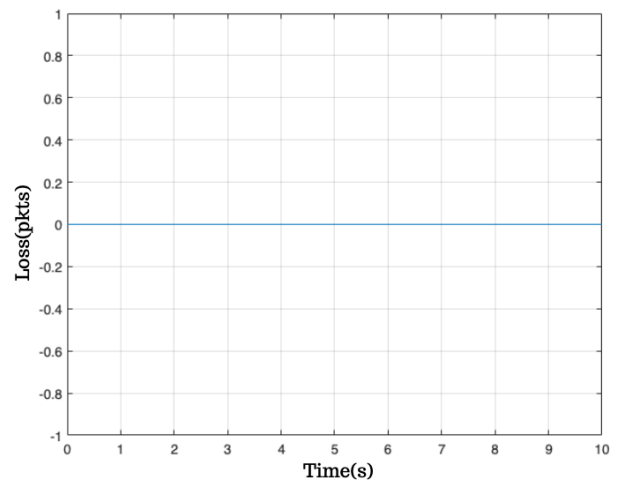
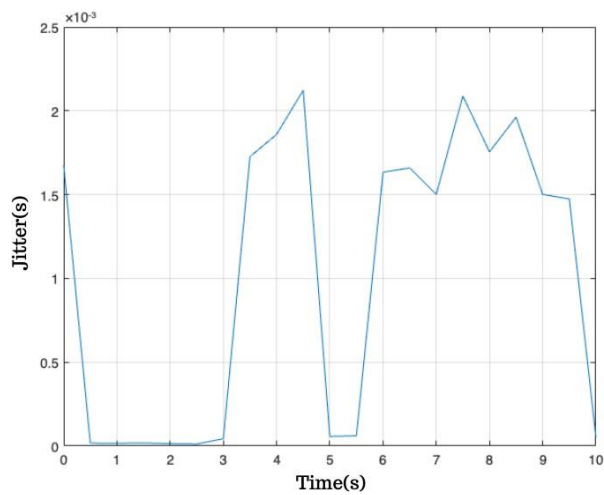
plot(x,y);
grid
figure(1);
```

**Image 7.19 Matlab script(2nd exp./pt.1.1.2)**

## DIAGRAMS

H1 -> H6





**Image 7.20 Metrics' diagrams(2nd exp./pt.1.1.2)**

We did the same process with the hosts 2,3,4 with the difference being that we used a different script:

```
-a 10.0.0.4 -rp 10001 -C 2000 -c 512 -T UDP
-a 10.0.0.4 -rp 10002 -C 3000 -c 256 -T UDP
-a 10.0.0.3 -rp 10003 -C 4000 -c 128 -T UDP|
```

**Image 7.21 Custom-Flow script (2nd exp./pt.1.2)**

```
"Node: h2"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend script_file2 -l sender.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Started sending packets of flow ID: 2
Started sending packets of flow ID: 1
Started sending packets of flow ID: 3
Finished sending packets of flow ID: 2
Finished sending packets of flow ID: 3
Finished sending packets of flow ID: 1
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 2
From 10.0.0.2:37342
To 10.0.0.4:10002
-----
Total time = 9.999742 s
Total packets = 25315
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 6480640
Average bitrate = 5184.645764 Kbit/s
Average packet rate = 2531.565314 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 1
From 10.0.0.2:47918
To 10.0.0.4:10001
-----
Total time = 9.999748 s
Total packets = 17776
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 9101312
Average bitrate = 7281.233087 Kbit/s
Average packet rate = 1777.644737 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 3
From 10.0.0.2:52291
To 10.0.0.3:10003
-----
Total time = 9.999933 s
Total packets = 32063
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 4104832
Average bitrate = 3283.885632 Kbit/s
Average packet rate = 3206.913562 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
***** TOTAL RESULTS *****
Number of flows = 3
Total time = 10.017592 s
Total packets = 75150
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 15686784
Average bitrate = 15721.763463 Kbit/s
Average packet rate = 7502.801072 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

```
"Node: h4"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv2_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 10002
Listening on UDP port : 10001
Finish on UDP port : 10002
Finish on UDP port : 10001
^CFinish with CTRL-C!
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv2_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 2
From 10.0.0.2:37342
To 10.0.0.4:10002
-----
Total time = 9.997750 s
Total packets = 25315
Minimum delay = 0.002014 s
Maximum delay = 0.007103 s
Average delay = 0.002061 s
Average jitter = 0.000011 s
Delay standard deviation = 0.000091 s
Bytes received = 6480640
Average bitrate = 5185.678778 Kbit/s
Average packet rate = 2532.063716 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 1
From 10.0.0.2:47918
To 10.0.0.4:10001
-----
Total time = 9.996937 s
Total packets = 17776
Minimum delay = 0.002014 s
Maximum delay = 0.009335 s
Average delay = 0.002053 s
Average jitter = 0.000015 s
Delay standard deviation = 0.000156 s
Bytes received = 9101312
Average bitrate = 7283.280463 Kbit/s
Average packet rate = 1778.144546 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
***** TOTAL RESULTS *****
Number of flows = 2
Total time = 10.000744 s
Total packets = 43091
Minimum delay = 0.002014 s
Maximum delay = 0.009335 s
Average delay = 0.002052 s
Average jitter = 0.000013 s
Delay standard deviation = 0.000122 s
Bytes received = 15591552
Average bitrate = 12464.534231 Kbit/s
Average packet rate = 4308.779427 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv2_log_file -c 500 s2.4
.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

```
"Node: h3"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv1_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 10003
Finish on UDP port : 10003
^CFinish with CTRL-C!
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv1_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 3
From 10.0.0.2:52291
To 10.0.0.3:10003
-----
Total time = 9.998946 s
Total packets = 32063
Minimum delay = 0.002015 s
Maximum delay = 0.010753 s
Average delay = 0.002043 s
Average jitter = 0.000003 s
Delay standard deviation = 0.000060 s
Bytes received = 4104832
Average bitrate = 3284.211756 Kbit/s
Average packet rate = 3207.238043 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
***** TOTAL RESULTS *****
Number of flows = 1
Total time = 9.998946 s
Total packets = 32063
Minimum delay = 0.002015 s
Maximum delay = 0.010753 s
Average delay = 0.002043 s
Average jitter = 0.000003 s
Delay standard deviation = 0.000060 s
Bytes received = 4104832
Average bitrate = 3284.211756 Kbit/s
Average packet rate = 3207.238043 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv1_log_file -c 500 s2.3
.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

Image 7.22 D-ITG Multi-flow traffic generation(2nd exp./pt.1.2)

The .dat files are shown below, as well as, the diagrams of each quality metric.

s2_3.dat					s2_4.dat				
0.000000	3295.232000	0.002057	0.000022	0.000000	0.000000	12472.320000	0.004210	0.000180	0.000000
0.500000	3289.088000	0.002047	0.000007	0.000000	0.500000	12500.992000	0.004093	0.000013	0.000000
1.000000	3291.136000	0.002047	0.000006	0.000000	1.000000	12476.416000	0.004094	0.000016	0.000000
1.500000	3282.944000	0.002047	0.000006	0.000000	1.500000	12492.800000	0.004094	0.000014	0.000000
2.000000	3299.328000	0.002049	0.000009	0.000000	2.000000	12492.800000	0.004100	0.000019	0.000000
2.500000	3287.040000	0.002047	0.000006	0.000000	2.500000	12435.456000	0.004103	0.000021	0.000000
3.000000	3274.752000	0.002052	0.000009	0.000000	3.000000	12460.032000	0.004099	0.000016	0.000000
3.500000	3287.040000	0.002052	0.000011	0.000000	3.500000	12435.456000	0.004098	0.000018	0.000000
4.000000	3282.944000	0.002050	0.000007	0.000000	4.000000	12439.552000	0.004099	0.000018	0.000000
4.500000	3282.944000	0.002048	0.000008	0.000000	4.500000	12492.800000	0.004095	0.000017	0.000000
5.000000	3229.696000	0.002053	0.000014	0.000000	5.000000	12353.536000	0.004129	0.000044	0.000000
5.500000	3287.040000	0.002048	0.000008	0.000000	5.500000	12455.936000	0.004097	0.000017	0.000000
6.000000	3287.040000	0.002046	0.000006	0.000000	6.000000	12509.184000	0.004094	0.000016	0.000000
6.500000	3289.088000	0.002046	0.000006	0.000000	6.500000	12484.608000	0.004092	0.000013	0.000000
7.000000	3282.944000	0.002055	0.000020	0.000000	7.000000	12435.456000	0.004105	0.000022	0.000000
7.500000	3284.992000	0.002050	0.000008	0.000000	7.500000	12464.128000	0.004103	0.000021	0.000000
8.000000	3295.232000	0.002046	0.000007	0.000000	8.000000	12496.896000	0.004093	0.000014	0.000000
8.500000	3289.088000	0.002045	0.000005	0.000000	8.500000	12480.512000	0.004092	0.000014	0.000000
9.000000	3287.040000	0.002047	0.000007	0.000000	9.000000	12468.224000	0.004094	0.000014	0.000000
9.500000	3272.704000	0.002048	0.000006	0.000000	9.500000	12447.744000	0.004096	0.000015	0.000000
					10.000000	12246.384000	0.002096	0.000024	0.000000

Image 7.23 dat Files(2nd exp./pt.1.2)

The first Matlab script :

```
load s2_3.dat
x=s2_3(:,1);
y=s2_3(:,5);

plot(x,y);
grid
figure(1);
```

Image 7.24 Matlab script(2nd exp./pt.1.2.1)

DIAGRAMS

H2 -> H3

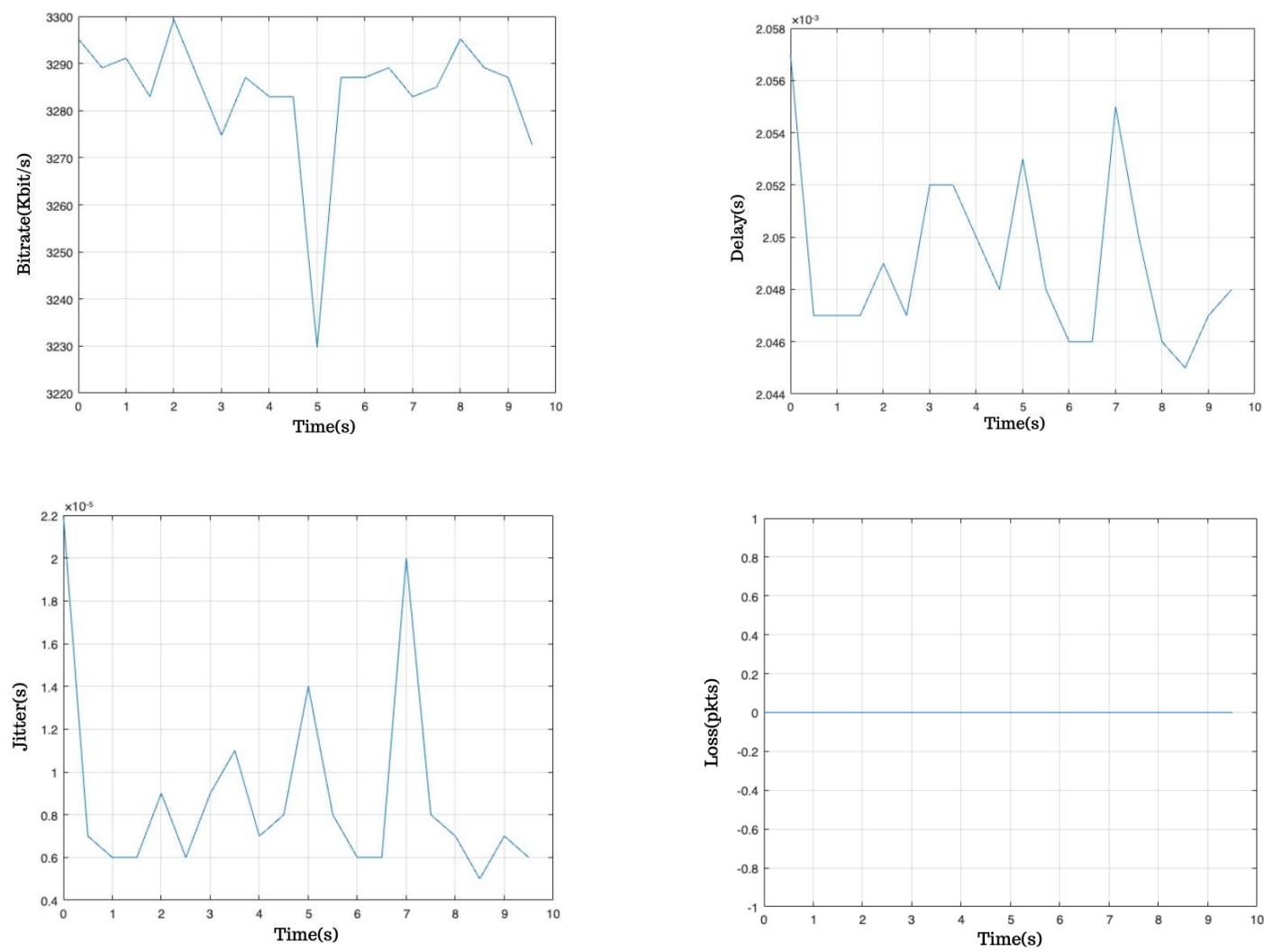


Image 7.25 Metrics’ diagrams(2nd exp./pt.1.2.1)

The second Matlab script :

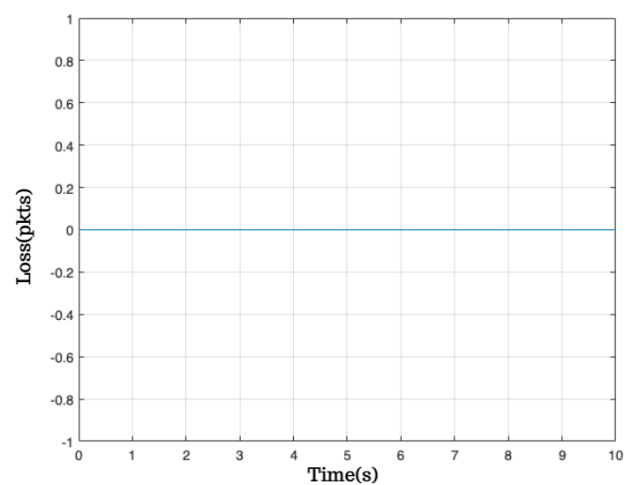
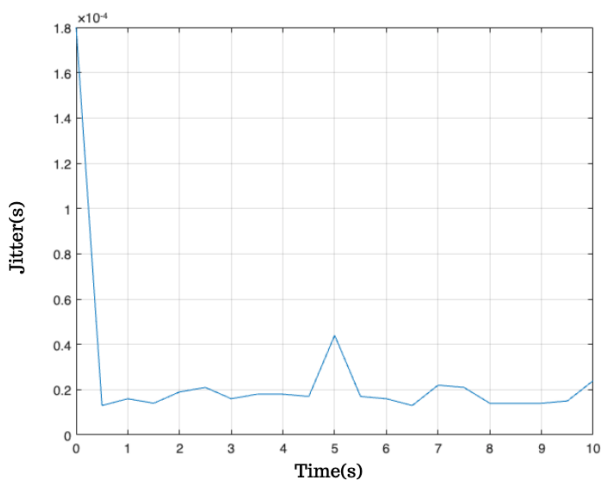
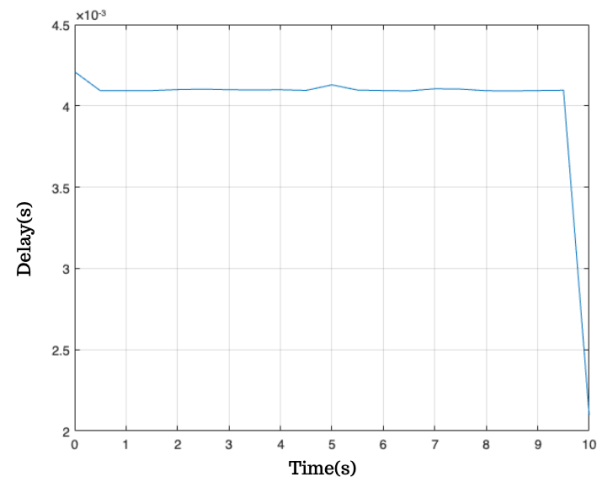
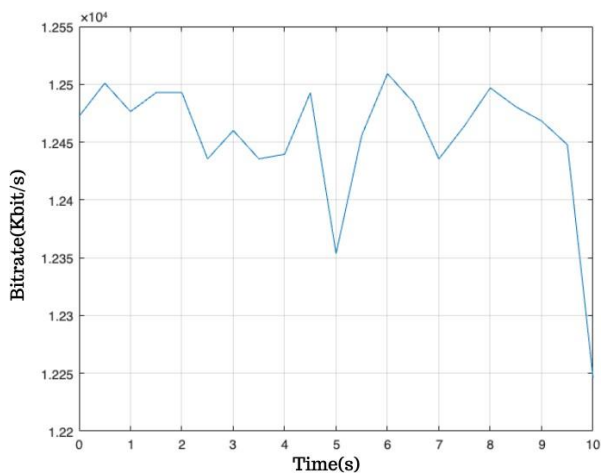
```
load s2_4.dat
x=s2_4(:,1);
y=s2_4(:,5);

plot(x,y);
grid
figure(1);
```

**Image 7.26 Matlab script(2nd exp./pt.1.2.2)**

## DIAGRAMS

H2 -> H4



**Image 7.27 Metrics' diagrams(2nd exp./pt.1.2.2)**

For the second part of this experiment we created a network of 10 hosts and connected it to POX, similarly to the one used before and then generated traffic between hosts 1 and 10, through multiple flows, with a script.

```
alekos@alekos-X555LB:~$ sudo mn --topo single,10 --controller=remote --link tc,delay=1ms
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Adding switches:
s1
*** Adding links:
(1ms delay) (1ms delay) (h1, s1) (1ms delay) (1ms delay) (h2, s1) (1ms delay) (1ms delay) (h3, s1) (1ms delay) (1ms delay) (h4, s1) (1ms delay) (1ms delay) (h5, s1) (1ms delay) (1ms delay) (h6, s1) (1ms delay) (1ms delay) (h7, s1) (1ms delay) (1ms delay) (h8, s1) (1ms delay) (1ms delay) (h9, s1) (1ms delay) (1ms delay) (h10, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Starting controller
c0
*** Starting 1 switches
s1 ...(1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay) (1ms delay)
*** Starting CLI:
mininet-wifi> xterm h1
mininet-wifi> xterm h10
mininet-wifi> exit

alekos@alekos-X555LB: ~/pox
File Edit View Search Terminal Help
alekos@alekos-X555LB:~$ cd pox
alekos@alekos-X555LB:~/pox$ ./pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Image 7.28 Ten Host topology and Pox activation(2nd exp./pt.2)



The script used is the following. As before we have VoIP type flow, a Telnet type and a DNS one. We also added a custom random flow in which the packet size is 512bytes and the rate is 2000pkts/s. Lastly, it is noteworthy that the last flow used emulates the transmission of a video file. With a rate of 270000pkts/s for 1080 resolution.

```
-a 10.0.0.10 -rp 10001 VoIP -x G.711.2 -h RTP -VAD
-a 10.0.0.10 -rp 10002 Telnet
-a 10.0.0.10 -rp 10003 DNS
-a 10.0.0.10 -rp 10004 -C 2000 -c 512 -T UDP
-a 10.0.0.10 -rp 10005 -C 270000 -c 1024 -T UDP
```

**Image 7.29 Custom-Flow script (2nd exp./pt.2)**

Using the same ITGSend and ITGRecv described before we begin the experiment.

```

"Node: h1"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend script_file -l sender.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Voice Codec: G.711
Framesize: 80.00
Samples: 2
Packets per sec.: 50
M/D: Si
Started sending packets of flow ID: 1
Started sending packets of flow ID: 3
Started sending packets of flow ID: 2
Started sending packets of flow ID: 4
Started sending packets of flow ID: 5
Finished sending packets of flow ID: 1
Finished sending packets of flow ID: 5
Finished sending packets of flow ID: 4
Finished sending packets of flow ID: 2
Finished sending packets of flow ID: 3
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 5
From 10.0.0.1:46503
To 10.0.0.10:10005
-----
Total time = 9.999997 s
Total packets = 731172
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 748720128
Average bitrate = 598976.282093 Kbit/s
Average packet rate = 73117.221935 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

"Node: h10"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on TCP port : 10002
Listening on UDP port : 10001
Listening on UDP port : 10005
Listening on UDP port : 10004
Listening on UDP port : 10003
Finish on UDP port : 10001
Finish on UDP port : 10005
Finish on UDP port : 10004
Finish on TCP port : 10002
Finish on UDP port : 10003
^CFinish with CTRL-C
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 5
From 10.0.0.1:46503
To 10.0.0.10:10005
-----
Total time = 9.996357 s
Total packets = 730015
Minimum delay = 0.002006 s
Maximum delay = 0.295476 s
Average delay = 0.002139 s
Average jitter = 0.000174 s
Delay standard deviation = 0.004643 s
Bytes received = 747535360
Average bitrate = 598246.229101 Kbit/s
Average packet rate = 73028.104138 pkt/s
Packets dropped = 1157 (0.16 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 4
From 10.0.0.1:41872
To 10.0.0.10:10004
-----
Total time = 9.996372 s
Total packets = 17886
Minimum delay = 0.002012 s
Maximum delay = 0.152175 s
Average delay = 0.002055 s
Average jitter = 0.000059 s
Delay standard deviation = 0.001682 s
Bytes received = 9157632
Average bitrate = 7328.764476 Kbit/s
Average packet rate = 1789.249140 pkt/s
Packets dropped = 5 (0.03 %)
Average loss-burst size = 0.000000 pkt
-----

```

\*\*

\*\*

```

Flow number: 4
From 10.0.0.1:41872
To 10.0.0.10:10004
-----
Total time = 9.999576 s
Total packets = 17831
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 9160132
Average bitrate = 7328.464327 Kbit/s
Average packet rate = 1783.175861 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

Flow number: 2
From 10.0.0.1:41774
To 10.0.0.10:10002
-----
Total time = 9.534990 s
Total packets = 1037
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 2171
Average bitrate = 1.821502 Kbit/s
Average packet rate = 108.757324 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

Flow number: 1
From 10.0.0.1:33424
To 10.0.0.10:10001
-----
Total time = 9.987828 s
Total packets = 499
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 57884
Average bitrate = 46.363634 Kbit/s
Average packet rate = 49.960812 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

Flow number: 3
From 10.0.0.1:52994
To 10.0.0.10:10003
-----
Total time = 8.935359 s
Total packets = 6
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 1344
Average bitrate = 1.203309 Kbit/s
Average packet rate = 0.671490 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

***** TOTAL RESULTS *****
-----
Number of flows = 5
Total time = 10.011716 s
Total packets = 750605
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 757941719
Average bitrate = 605643.802320 Kbit/s
Average packet rate = 74972.662029 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
-----

```

```
~root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

```

Flow number: 2
From 10.0.0.1:41774
To 10.0.0.10:10002
-----
Total time = 9.536402 s
Total packets = 1037
Minimum delay = 0.002021 s
Maximum delay = 0.013824 s
Average delay = 0.003568 s
Average jitter = 0.001638 s
Delay standard deviation = 0.001344 s
Bytes received = 2171
Average bitrate = 1.821232 Kbit/s
Average packet rate = 108.741221 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

Flow number: 1
From 10.0.0.1:33424
To 10.0.0.10:10001
-----
Total time = 9.982987 s
Total packets = 499
Minimum delay = 0.002014 s
Maximum delay = 0.006866 s
Average delay = 0.002037 s
Average jitter = 0.000018 s
Delay standard deviation = 0.000217 s
Bytes received = 57884
Average bitrate = 46.386070 Kbit/s
Average packet rate = 49.984989 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

Flow number: 3
From 10.0.0.1:52994
To 10.0.0.10:10003
-----
Total time = 8.933388 s
Total packets = 6
Minimum delay = 0.002023 s
Maximum delay = 0.004002 s
Average delay = 0.002375 s
Average jitter = 0.000399 s
Delay standard deviation = 0.000729 s
Bytes received = 1344
Average bitrate = 1.203575 Kbit/s
Average packet rate = 0.671638 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```
***** TOTAL RESULTS *****
```

```

-----
Number of flows = 5
Total time = 10.009697 s
Total packets = 749443
Minimum delay = 0.002006 s
Maximum delay = 0.295476 s
Average delay = 0.002139 s
Average jitter = 0.000173 s
Delay standard deviation = 0.004591 s
Bytes received = 756754391
Average bitrate = 604817.021734 Kbit/s
Average packet rate = 74871.696915 pkt/s
Packets dropped = 1162 (0.15 %)
Average loss-burst size = 0.000000 pkt
Error lines = 0
-----

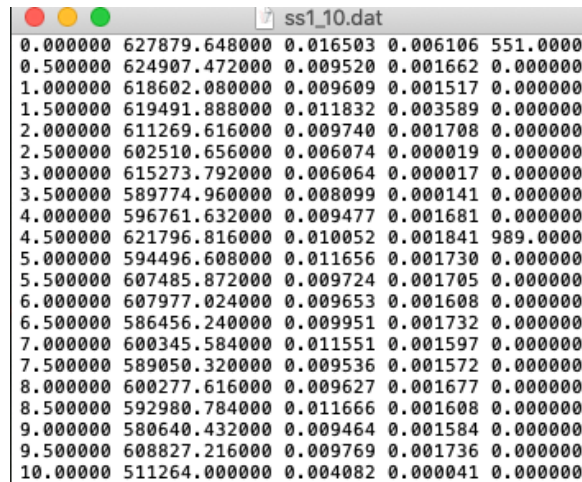
```

```

~root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv_log_file -c 500 ssl_1
*.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
~root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

Image 7.30 D-ITG Multi-flow traffic generation(2nd exp./pt2)

You can see that even though we added no losses to the network, due to the intensity and nature of the flows we have a slight loss of packets. It can be identified in the two last flows, with the majority of it being in the flow of the video transmission emulation.  
The .dat file is shown below, as well as, the diagrams of each quality metric.



0.000000	627879.648000	0.016503	0.006106	551.0000
0.500000	624907.472000	0.009520	0.001662	0.000000
1.000000	618602.080000	0.009609	0.001517	0.000000
1.500000	619491.888000	0.011832	0.003589	0.000000
2.000000	611269.616000	0.009740	0.001708	0.000000
2.500000	602510.656000	0.006074	0.000019	0.000000
3.000000	615273.792000	0.006064	0.000017	0.000000
3.500000	589774.960000	0.008099	0.000141	0.000000
4.000000	596761.632000	0.009477	0.001681	0.000000
4.500000	621796.816000	0.010052	0.001841	989.0000
5.000000	594496.608000	0.011656	0.001730	0.000000
5.500000	607485.872000	0.009724	0.001705	0.000000
6.000000	607977.024000	0.009653	0.001608	0.000000
6.500000	586456.240000	0.009951	0.001732	0.000000
7.000000	600345.584000	0.011551	0.001597	0.000000
7.500000	589050.320000	0.009536	0.001572	0.000000
8.000000	600277.616000	0.009627	0.001677	0.000000
8.500000	592980.784000	0.011666	0.001608	0.000000
9.000000	580640.432000	0.009464	0.001584	0.000000
9.500000	608827.216000	0.009769	0.001736	0.000000
10.000000	511264.000000	0.004082	0.000041	0.000000

**Image 7.31 dat Files(2nd exp./pt.2)**

The Matlab script:

```
load ss1_10.dat
x=ss1_10(:,1);
y=ss1_10(:,5);

plot(x,y);
grid
figure(1);
```

**Image 7.32 Matlab script(2nd exp./pt.2)**

DIAGRAMS

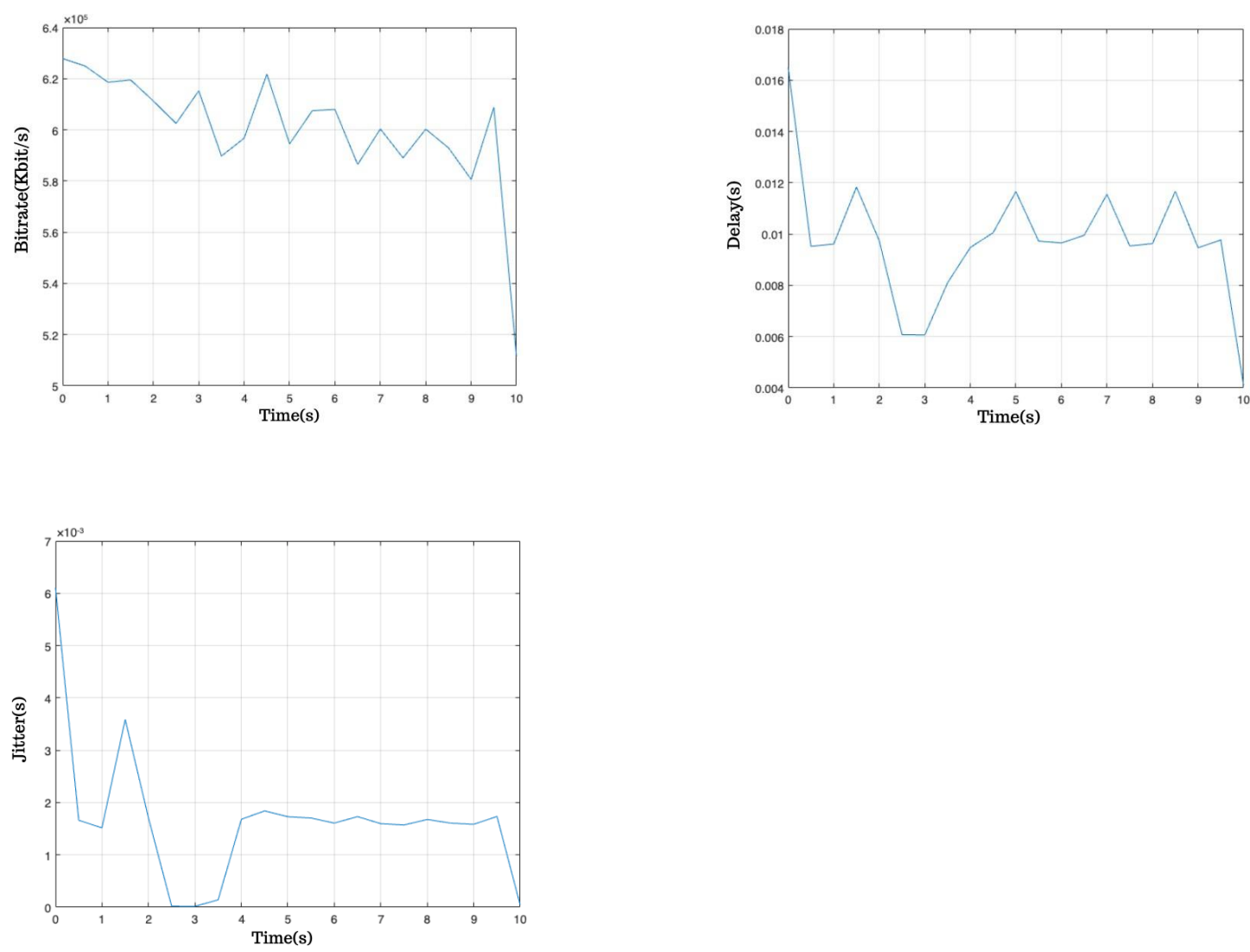


Image 7.33 Metrics' diagrams(2nd exp./pt.2)

### 7.3 Third Experiment: Multiple Flows with Loss Networks

In our third and final experiment we emulate networks similar to the one used in the last part of the second experiment, with the difference being that this time we add loss at the network's links. For starters, we create the network as shown before, with the addition of a loss factor, and connect it to POX.

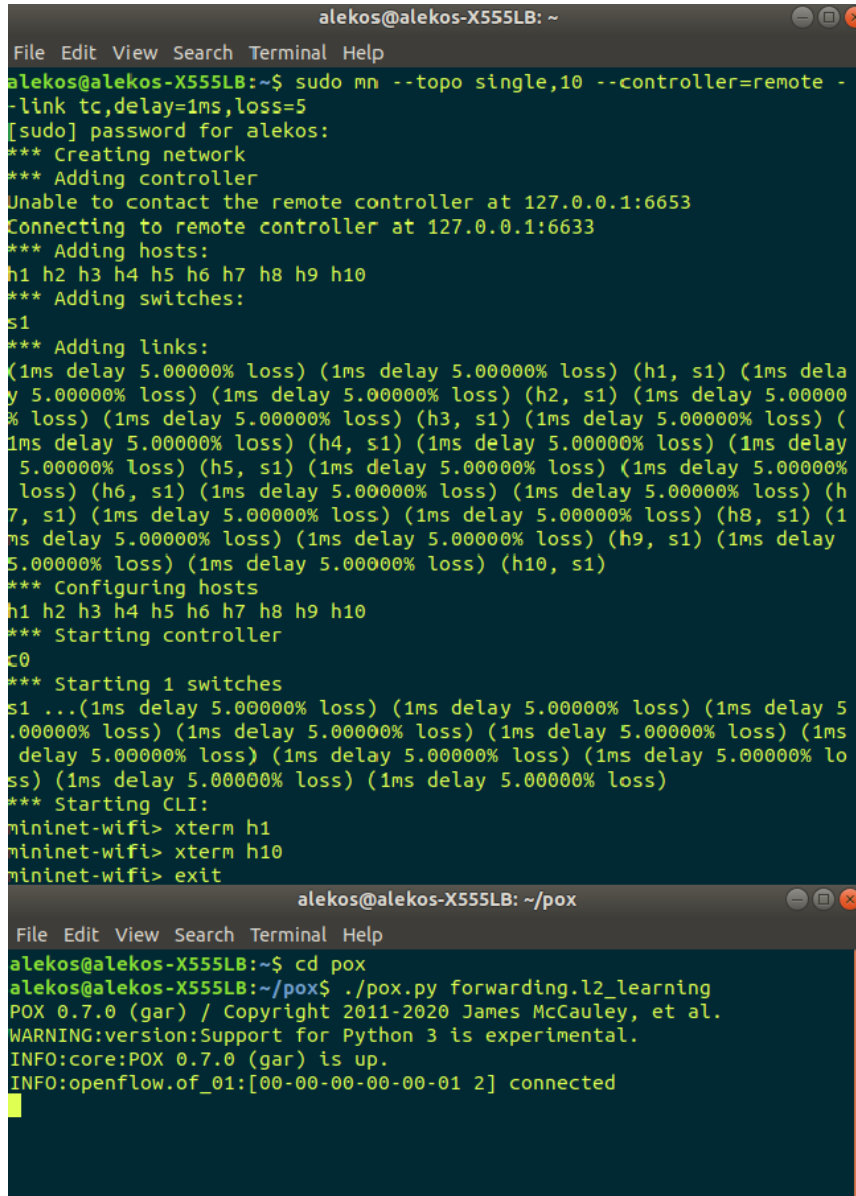
The image shows two terminal windows. The top window, titled 'alekos@alekos-X555LB: ~', shows the execution of the 'mn' command to create a network topology. The command is 'mn --topo single,10 --controller=remote -l tc,delay=1ms,loss=5'. The output shows the network being created with 10 hosts (h1-h10) and 1 switch (s1). Links are added between each host and the switch, each with a 1ms delay and 5.00000% loss. The controller is started, and the CLI is accessed. The bottom window, titled 'alekos@alekos-X555LB: ~/pox', shows the execution of 'pox.py forwarding.l2\_learning'. The output shows the POX version (0.7.0), a warning about Python 3 support, and the connection to the OpenFlow controller (INFO:openflow.of\_01:[00-00-00-00-00-01 2] connected).

Image 7.34 Ten Host topology and Pox activation(3rd exp./pt.1)

The process that follows is the same as before, with the difference being in the results.

```
"Node: h1"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend script_file -l sender.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Voice Codec: G.711
Framesize: 80,00
Samples: 2
Packets per sec.: 50
VAD: Si
Started sending packets of flow ID: 3
Started sending packets of flow ID: 5
Started sending packets of flow ID: 4
Started sending packets of flow ID: 1
Started sending packets of flow ID: 2
Finished sending packets of flow ID: 5
Finished sending packets of flow ID: 4
Finished sending packets of flow ID: 1
Finished sending packets of flow ID: 2
Finished sending packets of flow ID: 3
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 5
From 10.0.0.1:56853
To 10.0.0.10:10005
-----
Total time = 9.999983 s
Total packets = 750608
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 768622592
Average bitrate = 614898.504029 Kbit/s
Average packet rate = 75060.852543 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 4
From 10.0.0.1:45785
To 10.0.0.10:10004
-----
Total time = 9.999855 s
Total packets = 17877
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 5153024
Average bitrate = 7322.525377 Kbit/s
Average packet rate = 1787.725922 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 2
From 10.0.0.1:41956
To 10.0.0.10:10002
-----
Total time = 9.999496 s
Total packets = 1037
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 2195
Average bitrate = 1.756089 Kbit/s
Average packet rate = 103.705227 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 1
From 10.0.0.1:47632
To 10.0.0.10:10001
-----
Total time = 9.988052 s
Total packets = 499
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 57884
Average bitrate = 46.362594 Kbit/s
Average packet rate = 49.959692 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
"Node: h10"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 10003
Listening on UDP port : 10005
Listening on TCP port : 10002
Listening on UDP port : 10004
Listening on UDP port : 10001
Finish on UDP port : 10005
Finish on UDP port : 10004
Finish on UDP port : 10001
Finish on TCP port : 10002
Finish on UDP port : 10003
CFinish with CTRL-C
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 5
From 10.0.0.1:56853
To 10.0.0.10:10005
-----
Total time = 9.994276 s
Total packets = 675771
Minimum delay = 0.002007 s
Maximum delay = 0.434287 s
Average delay = 0.002264 s
Average jitter = 0.000377 s
Delay standard deviation = 0.008153 s
Bytes received = 631989504
Average bitrate = 553908.68517 Kbit/s
Average packet rate = 67615.803236 pkt/s
Packets dropped = 74837 (9.97 %)
Average loss-burst size = 1.189815 pkt
-----
Flow number: 4
From 10.0.0.1:45785
To 10.0.0.10:10004
-----
Total time = 9.835321 s
Total packets = 16130
Minimum delay = 0.002012 s
Maximum delay = 0.447912 s
Average delay = 0.006856 s
Average jitter = 0.004445 s
Delay standard deviation = 0.052817 s
Bytes received = 8266660
Average bitrate = 6717.470635 Kbit/s
Average packet rate = 1640.007479 pkt/s
Packets dropped = 1747 (9.77 %)
Average loss-burst size = 1.403213 pkt
-----
Flow number: 1
From 10.0.0.1:47632
To 10.0.0.10:10001
-----
Total time = 9.812237 s
Total packets = 448
Minimum delay = 0.002014 s
Maximum delay = 0.445315 s
Average delay = 0.010093 s
Average jitter = 0.013897 s
Delay standard deviation = 0.057462 s
Bytes received = 51968
Average bitrate = 42.363951 Kbit/s
Average packet rate = 45.657275 pkt/s
Packets dropped = 51 (10.22 %)
Average loss-burst size = 1.275000 pkt
-----
Flow number: 2
From 10.0.0.1:41956
To 10.0.0.10:10002
-----
Total time = 10.239788 s
Total packets = 1037
Minimum delay = 0.002025 s
Maximum delay = 1.658067 s
Average delay = 0.385784 s
Average jitter = 0.013244 s
Delay standard deviation = 0.427839 s
Bytes received = 2195
Average bitrate = 1.714879 Kbit/s
Average packet rate = 101.271628 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 3
From 10.0.0.1:53730
To 10.0.0.10:10003
-----
Total time = 8.927454 s
Total packets = 5
Minimum delay = 0.002027 s
Maximum delay = 0.007899 s
Average delay = 0.003227 s
Average jitter = 0.001468 s
Delay standard deviation = 0.002337 s
Bytes received = 909
Average bitrate = 0.814566 Kbit/s
Average packet rate = 0.580070 pkt/s
Packets dropped = 1 (16.67 %)
Average loss-burst size = 1.000000 pkt
-----
```

\*\*

\*\*

\*\*

\*\*

```

Flow number: 3
From 10.0.0.1:53730
To 10.0.0.10:10003
-----
Total time = 8.933326 s
Total packets = 6
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 1167
Average bitrate = 1.045075 Kbit/s
Average packet rate = 0.671642 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----

```

```

***** TOTAL RESULTS *****
Number of flows = 5
Total time = 10.485369 s
Total packets = 770027
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 777836862
Average bitrate = 593464.593616 Kbit/s
Average packet rate = 73438.235698 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
-----

```

```

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#

```

```

***** TOTAL RESULTS *****
Number of flows = 5
Total time = 10.719948 s
Total packets = 693391
Minimum delay = 0.002007 s
Maximum delay = 1.658067 s
Average delay = 0.002991 s
Average jitter = 0.000519 s
Delay standard deviation = 0.025023 s
Bytes received = 700303136
Average bitrate = 522616.815679 Kbit/s
Average packet rate = 64682.310026 pkt/s
Packets dropped = 76636 (9.95 %)
Average loss-burst size = 1.194005 pkt
Error lines = 0
-----

```

```

root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv_log_file -c 500 t1_10
.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#

```

**Image 7.35 D-ITG Multi-flow traffic generation(3rd exp./pt1)**

As it was expected, this time the loss percentage was significantly greater, as all links geared a default loss factor.

The .dat file is shown below, as well as, the diagrams of each quality metric.

```

t1_10.dat
0.000000 594472.336000 0.164451 0.199551 4468.000000
0.500000 569721.808000 0.158547 0.185211 3667.000000
1.000000 557342.592000 0.006071 0.000017 3691.000000
1.500000 560716.128000 0.008210 0.005778 3651.000000
2.000000 569539.376000 1.055431 0.016191 3809.000000
2.500000 566871.360000 0.223290 0.018297 3760.000000
3.000000 552362.992000 0.320104 0.014334 3734.000000
3.500000 539733.488000 0.010160 0.116204 3662.000000
4.000000 570595.776000 0.152533 0.010227 3746.000000
4.500000 556324.256000 0.188415 0.013315 3962.000000
5.000000 546129.600000 0.006075 0.000017 3621.000000
5.500000 563860.752000 0.587763 0.018252 3781.000000
6.000000 555595.936000 0.341808 0.009421 4082.000000
6.500000 557297.792000 0.280342 0.006069 3609.000000
7.000000 568530.448000 0.075820 0.012605 4824.000000
7.500000 560634.592000 0.110115 0.006234 3598.000000
8.000000 564071.280000 0.139548 0.013075 3813.000000
8.500000 564579.488000 0.072835 0.010681 3858.000000
9.000000 549881.904000 0.134632 0.027234 3680.000000
9.500000 536587.152000 0.088051 0.008495 3620.000000
10.000000 546587.675000 0.013690 0.034000 3709.000000
10.500000 556587.245000 0.308244 0.020867 3421.000000

```

**Image 7.36 dat Files(3rd exp./pt.1)**

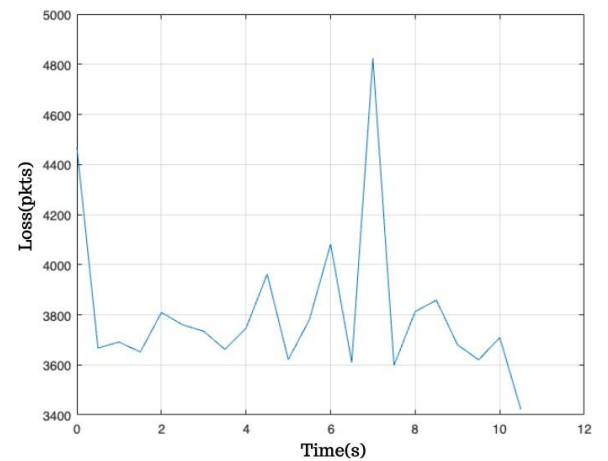
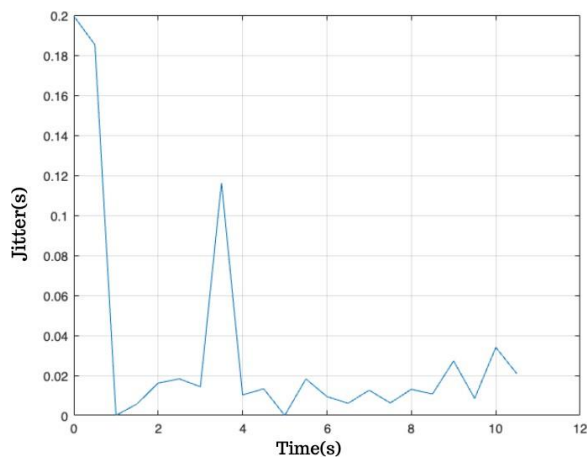
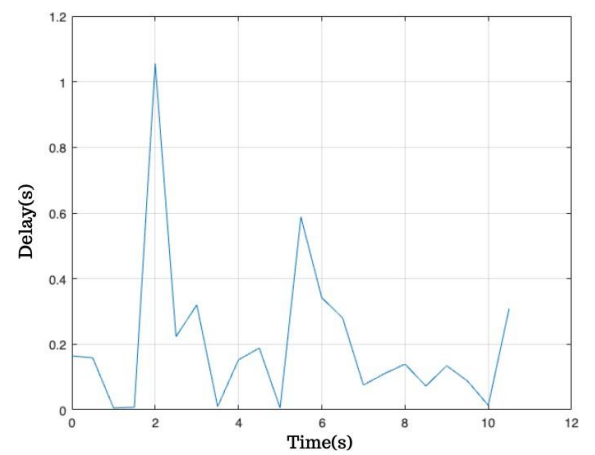
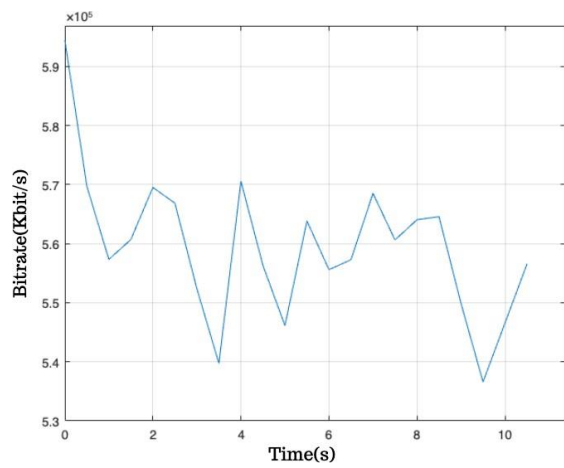
The Matlab script:

```
load t1_10.dat
x=t1_10(:,1);
y=t1_10(:,2);

plot(x,y);
grid
figure(1);
```

**Image 7.37 Matlab script(3rd exp./pt.1)**

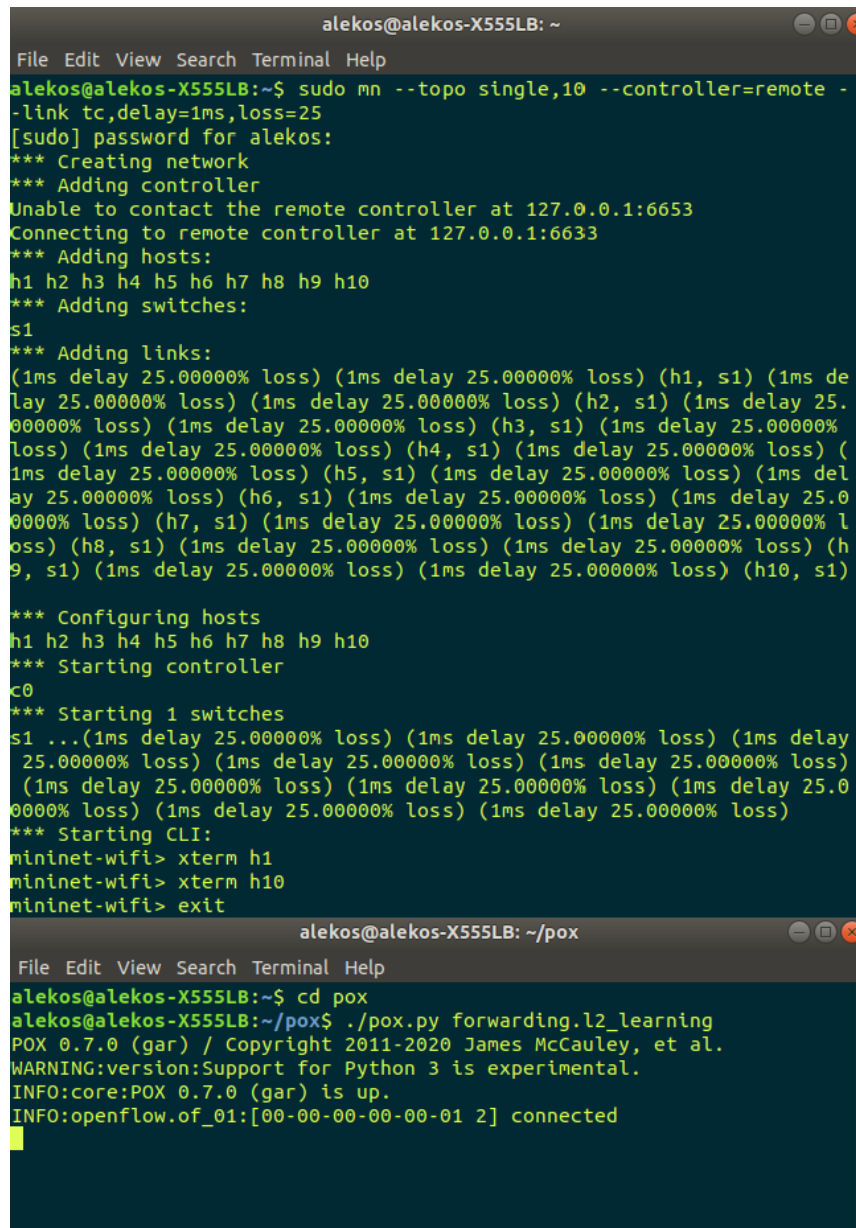
## DIAGRAMS



**Image 7.38 Metrics' diagrams(3rd exp./pt.1)**



In the second part of this experiment we simply increase the links' loss percentage to 25, in order to see how the network becomes truly insufficient to provide service under these conditions.



```
alekos@alekos-X555LB: ~  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ sudo mn --topo single,10 --controller=remote -  
-link tc,delay=1ms,loss=25  
[sudo] password for alekos:  
*** Creating network  
*** Adding controller  
Unable to contact the remote controller at 127.0.0.1:6653  
Connecting to remote controller at 127.0.0.1:6633  
*** Adding hosts:  
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10  
*** Adding switches:  
s1  
*** Adding links:  
(1ms delay 25.00000% loss) (1ms delay 25.00000% loss) (h1, s1) (1ms de  
lay 25.00000% loss) (1ms delay 25.00000% loss) (h2, s1) (1ms delay 25.  
00000% loss) (1ms delay 25.00000% loss) (h3, s1) (1ms delay 25.00000%  
loss) (1ms delay 25.00000% loss) (h4, s1) (1ms delay 25.00000% loss) (  
1ms delay 25.00000% loss) (h5, s1) (1ms delay 25.00000% loss) (1ms del  
ay 25.00000% loss) (h6, s1) (1ms delay 25.00000% loss) (1ms delay 25.0  
0000% loss) (h7, s1) (1ms delay 25.00000% loss) (1ms delay 25.00000% l  
oss) (h8, s1) (1ms delay 25.00000% loss) (1ms delay 25.00000% loss) (h  
9, s1) (1ms delay 25.00000% loss) (1ms delay 25.00000% loss) (h10, s1)  
  
*** Configuring hosts  
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...(1ms delay 25.00000% loss) (1ms delay 25.00000% loss) (1ms delay  
25.00000% loss) (1ms delay 25.00000% loss) (1ms delay 25.00000% loss)  
(1ms delay 25.00000% loss) (1ms delay 25.00000% loss) (1ms delay 25.0  
0000% loss) (1ms delay 25.00000% loss) (1ms delay 25.00000% loss)  
*** Starting CLI:  
mininet-wifi> xterm h1  
mininet-wifi> xterm h10  
mininet-wifi> exit  
  
alekos@alekos-X555LB: ~/pox  
File Edit View Search Terminal Help  
alekos@alekos-X555LB:~$ cd pox  
alekos@alekos-X555LB:~/pox$ ./pox.py forwarding.l2_learning  
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.  
WARNING:version:Support for Python 3 is experimental.  
INFO:core:POX 0.7.0 (gar) is up.  
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Image 7.39 Ten Host topology and Pox activation(3rd exp./pt.2)

```
"Node: h1"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGSend script_file -l sender.log
ITGSend version 2.8.1 (r1023)
Compile-time options: bursty multiport
Voice Codec: G.711
Framesize: 80.00
Samples: 2
Packets per sec.: 50
VAD: SI
Started sending packets of flow ID: 1
Started sending packets of flow ID: 3
Started sending packets of flow ID: 4
Started sending packets of flow ID: 5
Started sending packets of flow ID: 2
Finished sending packets of flow ID: 1
Finished sending packets of flow ID: 5
Finished sending packets of flow ID: 4
Finished sending packets of flow ID: 3
Finished sending packets of flow ID: 2
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 5
From 10.0.0.1:60308
To 10.0.0.10:10005
-----
Total time = 9.999934 s
Total packets = 858780
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 873390720
Average bitrate = 703512.998107 Kbit/s
Average packet rate = 85878.051527 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 4
From 10.0.0.1:46333
To 10.0.0.10:10004
-----
Total time = 9.999974 s
Total packets = 17862
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 9145856
Average bitrate = 7316.703823 Kbit/s
Average packet rate = 1786.304644 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 1
From 10.0.0.1:57391
To 10.0.0.10:10001
-----
Total time = 9.992133 s
Total packets = 493
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 57884
Average bitrate = 46.343659 Kbit/s
Average packet rate = 49.393287 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
Flow number: 2
From 10.0.0.1:41994
To 10.0.0.10:10002
-----
Total time = 7.606628 s
Total packets = 799
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 1742
Average bitrate = 1.833533 Kbit/s
Average packet rate = 105.122892 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
"Node: h10"
root@alekos-X555LB:~# cd D-ITG-2.8.1-r1023/bin
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGRecv -l recv_log_file
ITGRecv version 2.8.1 (r1023)
Compile-time options: bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 10001
Listening on TCP port : 10002
Listening on UDP port : 10003
Listening on UDP port : 10004
Listening on UDP port : 10005
Finish on UDP port : 10001
Finish on UDP port : 10005
Finish on UDP port : 10004
Finish on UDP port : 10003
Finish on TCP port : 10002
Ctrl-C with CTRL-C!
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv_log_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 5
From 10.0.0.1:60308
To 10.0.0.10:10005
-----
Total time = 9.994635 s
Total packets = 482215
Minimum delay = 0.002006 s
Maximum delay = 0.323074 s
Average delay = 0.002177 s
Average jitter = 0.000249 s
Delay standard deviation = 0.005777 s
Bytes received = 493788160
Average bitrate = 395242.575642 Kbit/s
Average packet rate = 48247.384722 pkt/s
Packets dropped = 376565 (43.85 %)
Average loss-burst size = 1.798554 pkt
-----
Flow number: 4
From 10.0.0.1:46333
To 10.0.0.10:10004
-----
Total time = 9.966200 s
Total packets = 10086
Minimum delay = 0.002012 s
Maximum delay = 0.324084 s
Average delay = 0.002790 s
Average jitter = 0.001151 s
Delay standard deviation = 0.015409 s
Bytes received = 5164032
Average bitrate = 4145.236499 Kbit/s
Average packet rate = 1012.020630 pkt/s
Packets dropped = 7777 (45.54 %)
Average loss-burst size = 1.784944 pkt
-----
Flow number: 1
From 10.0.0.1:57391
To 10.0.0.10:10001
-----
Total time = 9.970816 s
Total packets = 267
Minimum delay = 0.002015 s
Maximum delay = 0.003177 s
Average delay = 0.002040 s
Average jitter = 0.000011 s
Delay standard deviation = 0.000084 s
Bytes received = 30972
Average bitrate = 24.849873 Kbit/s
Average packet rate = 26.777881 pkt/s
Packets dropped = 231 (46.39 %)
Average loss-burst size = 1.818898 pkt
-----
Flow number: 3
From 10.0.0.1:55175
To 10.0.0.10:10003
-----
Total time = 8.931523 s
Total packets = 4
Minimum delay = 0.002038 s
Maximum delay = 0.006176 s
Average delay = 0.003094 s
Average jitter = 0.001379 s
Delay standard deviation = 0.001780 s
Bytes received = 1105
Average bitrate = 0.989753 Kbit/s
Average packet rate = 0.447852 pkt/s
Packets dropped = 2 (33.33 %)
Average loss-burst size = 1.000000 pkt
-----
Flow number: 2
From 10.0.0.1:41994
To 10.0.0.10:10002
-----
Total time = 15.445838 s
Total packets = 168
Minimum delay = 0.002120 s
Maximum delay = 14.941860 s
Average delay = 10.358876 s
Average jitter = 0.105605 s
Delay standard deviation = 5.652699 s
Bytes received = 408
Average bitrate = 0.211319 Kbit/s
Average packet rate = 10.876716 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
```

\*\*

\*\*

\* \*

```
***** TOTAL RESULTS *****
Number of flows      = 5
total time           = 16.067407 s
total packets        = 492740
minimum delay        = 0.002006 s
maximum delay        = 14.941860 s
average delay        = 0.005721 s
average jitter       = 0.000303 s
delay standard deviation = 0.217922 s
bytes received       = 498984677
average bitrate      = 248445.652494 Kbit/s
average packet rate  = 30667.051628 pkt/s
packets dropped      = 384575 (43.84 %)
average loss-burst size = 1.798281 pkt
error lines          = 0
```

```
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin# ./ITGDec recv_log_file -c 500 ts1_1.dat
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
root@alekos-X555LB:~/D-ITG-2.8.1-r1023/bin#
```

**Image 7.40 D-ITG Multi-flow traffic generation(3rd exp./pt2)**

It is quit obvious that a data transmission, whatever the type may be, cannot operate with a loss factor of 43.84%. The service is inexistent.

The .dat file is shown below, as well as, the diagrams of each quality metric.

			ts1_10.dat	
0.000000	17284.4160000	0.018923	0.000869	1602.000000
0.500000	143061.088000	0.026479	0.027616	19175.000000
1.000000	404851.444000	0.311741	0.020256	19374.000000
1.500000	143853.056000	0.006127	0.000031	19925.000000
2.000000	397196.864000	0.006125	0.000055	19304.000000
2.500000	387224.576000	0.006070	0.000019	18544.000000
3.000000	381944.128000	0.006074	0.000019	18447.000000
3.500000	402015.616000	0.006066	0.000013	19493.000000
4.000000	402639.376000	0.008186	0.004066	19477.000000
4.500000	403590.336000	0.006067	0.000016	19421.000000
5.000000	400006.720000	0.006064	0.000016	19525.000000
5.500000	405820.096000	0.008192	0.000102	19168.000000
6.000000	401253.768000	0.006065	0.000019	19291.000000
6.500000	405475.264000	0.006063	0.000014	19615.000000
7.000000	393887.296000	0.006069	0.000015	18881.000000
7.500000	401131.968000	0.006063	0.000013	19434.000000
8.000000	394478.976000	0.006065	0.000014	18895.000000
8.500000	395992.640000	0.006074	0.000018	19035.000000
9.000000	392525.184000	0.008115	0.000020	18858.000000
9.500000	403563.904000	0.006056	0.000014	19513.000000
10.0000	365953.024000	0.004046	0.000010	17598.000000
10.5000	0.000000000000	0.000000	0.000000	0.000000000000
11.0000	0.000000000000	0.000000	0.000000	0.000000000000
11.5000	0.000000000000	0.000000	0.000000	0.000000000000
12.0000	0.000000000000	0.000000	0.000000	0.000000000000
12.5000	0.000000000000	0.000000	0.000000	0.000000000000
13.0000	0.000000000000	0.000000	0.000000	0.000000000000
13.5000	0.000000000000	0.000000	0.000000	0.000000000000
14.0000	0.000000000000	0.000000	0.000000	0.000000000000
14.5000	0.000000000000	0.000000	0.000000	0.000000000000
15.0000	0.000000000000	0.000000	0.000000	0.000000000000
15.5000	0.000000000000	0.000000	0.000000	0.000000000000
16.0000	5.104000000000	13.50287	0.131615	0.000000000000

**Image 7.41 dat Files(3rd exp./pt.2)**

The Matlab script:.

```
load ts1_10.dat
x=ts1_10(:,1);
y=ts1_10(:,3);

plot(x,y);
grid
figure(1);
```

Image 7.42 Matlab script(3rd exp./pt.2)

## DIAGRAMS

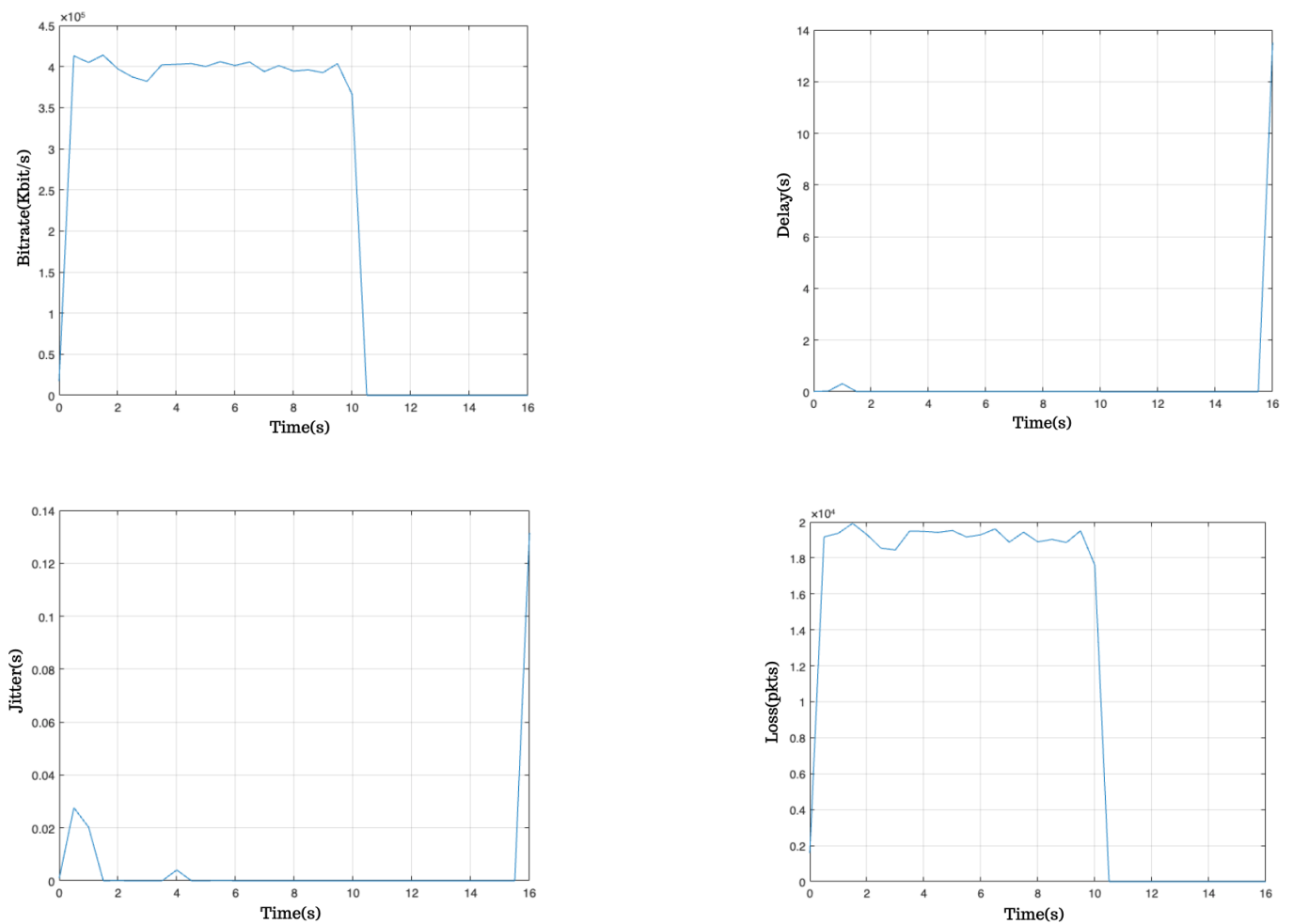


Image 7.43 Metrics' diagrams(3rd exp./pt.2)

## 8. Conclusion

In this thesis, the traditional networking architecture was presented, as well as, the reasons that make it obsolete. With the unprecedented rise of technology and the never ending series of demands that arise, it became clear that a static model with as many limitations as traditional networking, is not the one that can take us to the next era and thus came the rise of the revolutionary Software Defined Networks(SDN). With its ability to decouple the network control and forwarding functions , thus enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services, it became clear that this is the future.

Of course in order for anything to be proved successful, experiments must be conducted and results must be taken, and that is exactly what is done in this thesis. Taking advantage of SDN's easily programmable elements and through the use of programs such as the Mininet network emulator, POX and D-ITG, various virtual networks were tested and multiple performance metrics were taken and presented.

## Appendix

### D-ITG command options:

#### ITGSend:

##### Log options (log\_opts):

###### -l [logfile]

Generate sender-side log file (default: /tmp/ITGSend.log).

Generates a log file that contains timing, ordering and size information about every packet sent by ITGSend. Under Windows the default log file name is "ITGSend.log".

###### -x [receiver\_logfile]

Ask ITGRecv to generate receiver-side log file (default: /tmp/ITGRecv.log).

Generates a log file that contains timing, ordering and size information about every packet received by ITGRecv. Under Windows the default log file name is "ITGRecv.log".

##### Signaling options (sig\_opts):

###### -Sda <signaling\_dest\_addr>

Set the destination address for the signaling channel (default: equal to -a ).

###### -Sdp <signaling\_dest\_port>

Set the destination port for the signaling channel (default: 9000).

###### -Ssa <signaling\_src\_addr>

Set the source address for the signaling channel (default: Set by O.S.).

###### -Ssp <signaling\_src\_port>

Set the source port for the signaling channel (default: Set by O.S.).

##### Flow options (flow\_opts):

###### -t <duration>

Set the generation duration in ms (default: 10000 ms).

When -t, -z, and/or -k are specified, the most restrictive applies.

###### -z <#\_of\_packets>

Set the number of packets to generate.

When -t, -z, and/or -k are specified, the most restrictive applies.

###### -k <#\_of\_KBytes>

Set the number of KBytes to generate.

When -t, -z, and/or -k are specified, the most restrictive applies.

###### -d <delay>

Start the generation after the specified delay in ms (default: 0 ms).

-a <dest\_address>

Set the destination address (default: 127.0.0.1).

This option applies to both traffic flow and signaling channel unless

-Sda option is also specified.

-sa <src\_address>

Set the source address (default: Set by O.S.).

-rp <dest\_port>

Set the destination port (default: 8999).

This option applies only to traffic flows (i.e. not to signaling channel). See above for signaling channel.

-sp <src\_port>

Set the source port (default: Set by O.S.).

This option applies only to traffic flow. See above for signaling channel.

-T <protocol>

Layer 4 protocol (default: UDP):

- UDP (User Datagram Protocol)

- TCP (Transport Control Protocol)

- DCCP (Datagram Congestion Control Protocol)(DCCP protocol is currently supported only under Linux).

Inter-departure time options (idt\_opts):

-C <rate>

Constant (default: 1000 pkts/s).

-U <min\_rate><max\_rate>

Uniform distribution.

-E <mean\_rate>

Exponential distribution.

-N <mean><std\_dev>

Normal distribution.

Packet size options (ps\_opts):

-c <pkt\_size>

Constant (default: 512 bytes).

-u <min\_pkt\_size><max\_pkt\_size>

Uniform distribution.

-e <average\_pkt\_size>

Exponential distribution.

-n <mean><std\_dev>

Normal distribution.

Application layer options (app\_opts):

-Fp <filename>

Read payload content from file.

-Telnet

Emulate Telnet traffic. Generates traffic with Telnet characteristics. It works with TCP transport layer protocol. Different settings will be ignored.

-DNS

Emulate DNS traffic. Generates traffic with DNS characteristics. It works with both UDP and TCP transport layer protocols.

-VoIP

Emulate Voice-over-IP traffic.

-x<codec> VoIP sub-option: audio codec (default: G.711.1):

- G.711.<1 or 2> (samples per pkt)

- G.729.<2 or 3> (samples per pkt)

- G.723.1

-h

VoIP sub-option: audio transfer protocol (default: RTP): -

-RTP: Real Time Protocol (default)

- CRTP: Real Time Protocol with header compression

Misc options (misc\_opts):

-h | --help

Display this help and exit.

-s <seed>

Set the seed used for generating distributions (default: random). Sets the seed for the random number generator (allowed range [0,1]).

ITGRecv:

-h | --help

Display this help and exit

-l [logfile]

Enable logging to file (default filename: /tmp/ITGRecv.log). Generates a log file containing timing, ordering and size information about every received packet. If the -x option is set at the ITGSend side then this is overridden.

ITGLog:



-h | --help

Display this help and exit

-q <log\_buffer\_size>

Number of packets to push to the log at once (default: 50)

ITGDec:

-v

Print standard summary to stdout (default). Prints to stdout a summary comprising a set of statistics about each flow and the whole set of flows.

-l Print to the decoded log in text format. Generates the output log file in text format. For each packet the log file contains a line with the following fields:

- Flow: the flow number
- Seq: the sequence number of the packet
- Src: the source IP address and port number (e.g. 127.0.0.1/44225)
- Dest: the destination IP address and port number (e.g. 127.0.0.1/8999)
- txTime: the transmission time (e.g. 8:23:52.874105)
- rxTime: the reception time (e.g. 8:23:52.874105)
- Size: the size of the packet payload in bytes

-o <outfile>

Print to the decoded log for Octave/Matlab import. Generates the output log file in a format that can be directly imported from Octave/Matlab for further analysis. The output log content is the same as the one generated by the -l option.

-c [filename]

Print all average metrics to file every milliseconds (default filename: 'combined\_stats.dat'). Dumps to file all the average metrics as sampled every milliseconds. Each line of the output file respectively contains the following fields: - "Time", "Bitrate", "Delay", "Jitter", "Packet loss"

-h | --help

Display this help and exit.

[16]

## REFERENCES

- [1] <https://www.ibm.com/services/network/sdn-versus-traditional-networking>
- [2] [https://www.researchgate.net/publication/323974224\\_A\\_survey\\_and\\_classification\\_of\\_controller\\_placement\\_problem\\_in\\_SDN](https://www.researchgate.net/publication/323974224_A_survey_and_classification_of_controller_placement_problem_in_SDN)
- [3] <https://www.bitsinflight.com/sdns-promise-lives-on/>
- [4] <https://opennetworking.org/sdn-definition/>
- [5] <https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>
- [6] <https://www.techtarget.com/searchnetworking/definition/software-defined-networking-SDN>
- [7] <https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/>
- [8] <https://www.sciencedirect.com/topics/computer-science/openflow-protocol>
- [9] <https://medium.com/@AriaZhu/openflow-switch-what-is-it-and-how-does-it-work-7589ea7ea29c>
- [10] <https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf>
- [11] <https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf>
- [12] <https://www.brianlinkletter.com/2015/04/using-the-pox-sdn-controller/>
- [13] <https://noxrepo.github.io/pox-doc/html/#>
- [14] <https://noxrepo.github.io/pox-doc/html/#ofp-flow-mod-flow-table-modification>
- [15] <https://www.brianlinkletter.com/2015/09/using-pox-components-to-create-a-software-defined-networking-application/>
- [16] <http://traffic.comics.unina.it/software/ITG/manual/D-ITG-2.8.1-manual.pdf>
- [17] [https://www.researchgate.net/publication/221406698\\_D-ITG\\_Distributed\\_Internet\\_Traffic\\_Generator](https://www.researchgate.net/publication/221406698_D-ITG_Distributed_Internet_Traffic_Generator)
- [18] <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>
- [19] <http://mininet.org/walkthrough/>
- [20] <https://www.brianlinkletter.com/2015/04/how-to-use-miniedit-mininets-graphical-user-interface/>

[21] <https://usermanual.wiki/Pdf/mininetwifidraftmanual.297704656/view>