



**University of Piraeus**  
**Department of Digital Systems**

**UNIVERSITY OF PIRAEUS**  
**DEPARTMENT OF DIGITAL SYSTEMS**  
**POSTGRADUATE PROGRAMME**  
**“DIGITAL COMMUNICATIONS & NETWORKS”**

**DIPLOMA THESIS:**  
**DESIGN AND IMPLEMENTATION OF AN**  
**ANALYTICS**  
**APPLICATION IN SOFTWARE DEFINED**  
**NETWORKS**

**SUBMITTED BY:**  
**LAZARIS PANAGIOTIS**

**SUPERVISOR:**  
**PROF. TSAGKARIS KONSTANTINOS**

## Table of Contents

1. Abstract.....	3
2. Introduction.....	4
3. Software Defined Networks.....	6
3.1. SDN Architecture.....	7
4. OpenFlow.....	9
5. Analytics.....	16
5.1. SDN Analytics.....	17
6. Software Defined Network Analytics Application.....	17
6.1. Application Development Environment.....	18
6.2. Environment setup.....	19
6.2.1. Mininet.....	19
6.2.2. OpenDaylight Controller.....	21
6.3. SDN Analytics Application.....	25
6.3.1. Application Environment Setup.....	25
6.3.2. Application Development.....	29
6.3.3. Application scenario.....	30
6.3.3.1. Linear Network Topology.....	31
6.3.3.2. Ring Network Topology.....	33
6.3.3.3. Tree Network Topology.....	37

6.3.4. Results Analysis.....	<b>34</b>
6.3.4.1. Linear Topology Analysis.....	<b>40</b>
6.3.4.2. Ring Topology Analysis.....	<b>45</b>
6.3.4.3. Tree Topology Analysis.....	<b>51</b>
6.3.4.4. Topology Analysis Without Performance Parameters.....	<b>56</b>
6.3.4.5. Topology Analysis With the first set of Performance Parameters.....	<b>60</b>
6.3.4.6. Topology Analysis With the second set of Performance Parameters.....	<b>65</b>
7. Conclusion.....	<b>71</b>
8. References.....	<b>72</b>

## 1. *Abstract*

At the beginning of the 21<sup>st</sup> century, enterprise needs and networking requirements had already increased dramatically and network engineers should adapt immediately and find an efficient solution to cover the requirements. In addition, traditional network architectures could not cover the storage needs of data centres, the extended use of mobile devices, virtualization and cloud services, as the end users need to connect and have access to the network data and applications from everywhere made the majority of network architectures that enterprises were using ineffective.

Software Defined Networking (SDN) enabled engineers to create dynamic and programmable networks. Networks became flexible and more adaptable to storage needs, virtualization and Cloud services. Furthermore, Software Defined Networking provides centralized and fully programmable network control using Software Defined Network Controllers, software that allows network engineers control and manage network behaviour easily.

This study makes an extensive reference to Software Defined Network architecture, including OpenDaylight Controller and OpenFlow protocol that is used from the controller to interact with network elements. Finally, a Software Defined Network analytics application will be deployed and presented extensively. We are going to create a network topology with mininet, then OpenDaylight controller will be used to manage the network traffic and the deployed application will interact with OpenDaylight controller and present graphically the network traffic statistics.

## 2. Introduction

The first kind of a computer network was invented by George Stibitz in 1940. It was including a teletype used to send calculation messages to the Complex Number Computer over telegraph lines. When the calculations were complete George Stibitz was receiving the results by the same means. It was the first remote computer network.

In the late 1950s, a military radar system network named SAGE (Semi-Automatic Ground Environment) was created. In 1962, J.C.R Licklider developed a computer network (ARPANET) which allowed to the connected computers have access to each other stored programs and data. In 1964, American Airlines implemented the first online flight reservation system called SABRE (Semi-Automatic Business Research Environment) which was using telephone lines and could deliver data in less than three seconds. The first wide area network (WAN) was implemented in 1965 by Thomas Marill and Lawrence G.Roberts. In 1980s, TCP/IP internet protocol was introduced as networking protocol to ARPANET due to the expanded use. At the end of the same decade, appeared the first internet service providers (ISPs). In 2000, asymmetric digital subscriber lines were launched in UK and six years later there where around thirteen million homes using ADSL broadband connections. In 2005 and 2006, Cloud storage services were developed by Box and Amazon Web Services to cover the enterprises remote file sharing and data storage needs. The last ten years, the increased needs in remote access, data storage, virtualization, bandwidth and faster speed led the engineers to many important innovations such as 100 Gigabit Ethernet standard in 2010, fiber-optic broadband standard that offers broadband speed up to 100Mbps and the new 802.11ac Wi-Fi standard that offers speed up to 2Gbps.

In the next years, the demands will grow if we consider that apart from enterprise computer networks and telecommunication networks, internet of things (IoT) is almost here that includes new devises, features and demanding

applications as “Smart home”. As a result, speed bandwidth and connectivity demands are going to increase further in order to offer the needed quality of service for all of these applications and services. In order to control and manage all these new features, Internet of Things, Cloud services, virtualization and all the networking demands, network engineers could not depend on the classic network architectures and the introduction of a new technology was necessary. Software Defined Networking (SDN) introduced to make computer networks more programmable. It allows the control and management of complex networks that contain plenty of devices like routers and switches, middleboxes such as network address translators (NAT), firewalls. In addition, Software Defined Networking overcomes the complexity caused by different network protocols and configuration interfaces that different vendors use for their devices. SDN Control layer includes a single software control program that allows administrators to control and configure multiple network elements using an Application Programming Interface(API). OpenFlow is the most usual communication protocol and over the past few years many switches, from different vendors support it. Furthermore, developers have used SDN controller platforms to implement applications for network management such as network virtualization, traffic management, load balancing and network monitoring. In this study, OpenDaylight controller will be used combined with mininet, which is a virtual network emulator, to develop a Software Defined Networks analytics web application and every related technology and tool will be presented extensively.

### **3. Software Defined Networks (SDN)**

Software defined networking was introduced when network engineers and administrators realized that the architecture of traditional networks was not able to support the needs of more demanding environments anymore. The storage needs increased and continue to increase really fast, cloud services, network virtualization and the need to reduce operational costs demanded more dynamic and flexible networking. Software defined networking offers dynamic architecture, cost-effectiveness, programmability, supports high bandwidths and provides central management which is one of its biggest advantages. In contrast with traditional network architectures, Software defined networking separates the network controlling system that makes network decisions from the network elements that execute the controlling system's decisions and forwarding functions. This control separation allows networks become programmable, network administrators are able to easily adjust network traffic rules to avoid collisions and achieve the needed quality of service. Also developers use API provided by controllers to implement network services and applications. In addition, network elements can be easily managed and configured because their behaviour is defined by the controller and is independent from multiple protocols and rules that different vendors use. Finally, another advantage of Software Defined Networking is cost-effectiveness as the extended use of virtual machines, virtualization technologies and open standards managed to reduce the operational costs.

### 3.1. Software Defined Network Architecture

Software Defined Networking architecture is based on the separation of the control plane from the data layer of the network. SDN architecture generally includes three layers of functionality:

- Infrastructure or data layer: It includes all the networking devices and is responsible for the traffic forwarding in the data path based on the rules determined from the controller. In addition, data layer is responsible for managing the network devices, the number of available ports and their status, their memory and generally whatever relates to the state of network resources.
- Control layer: The main element of this layer is the controller. It defines the traffic control rules and relays them to the network equipment of the data layer such as SDN switches, routers and other network devices for execution. All of the management control and configuration functions are determined in this layer and SDN switches of the data layer simply manage flow tables. As a result, SDN Controllers enable centralized and functional network management, configuration and control. In addition, SDN Controller provides network information such as topology details and network statistics to the applications of the above layer through REST APIs. Controller is also responsible for the communication between the implemented application and services of the application layer and the elements of data layer. Finally, Control layer uses two different interfaces to communicate with data layer and application layer:
  - Northbound Interface: is used by Controllers to communicate with application layer. It is achieved through REST APIs generated by the Controller.



- Southbound Interface: is used by Controllers to communicate with the infrastructure layer network elements. It is achieved through southbound protocols such as Openflow and Netconf. It supports both physical and virtualized architectures.

Traditional network architectures are not suitable for larger and more complex networks, SDN architectures avoid this limitation with the use of more than one controller that is responsible for network elements with the same characteristics and a centralized Controller is responsible for their management. When more than one controllers are included in the control layers, they use westbound and eastbound interfaces to communicate.

- Application layer: It includes all the implemented applications that use northbound interface to communicate with the SDN controller. They get the required information and resources for decision-making purposes from the controller through APIs. On the other hand, SDN applications define network behaviour to the controller. Different types of SDN applications have been implemented, including network management, security, troubleshooting, network analytics and monitoring applications.

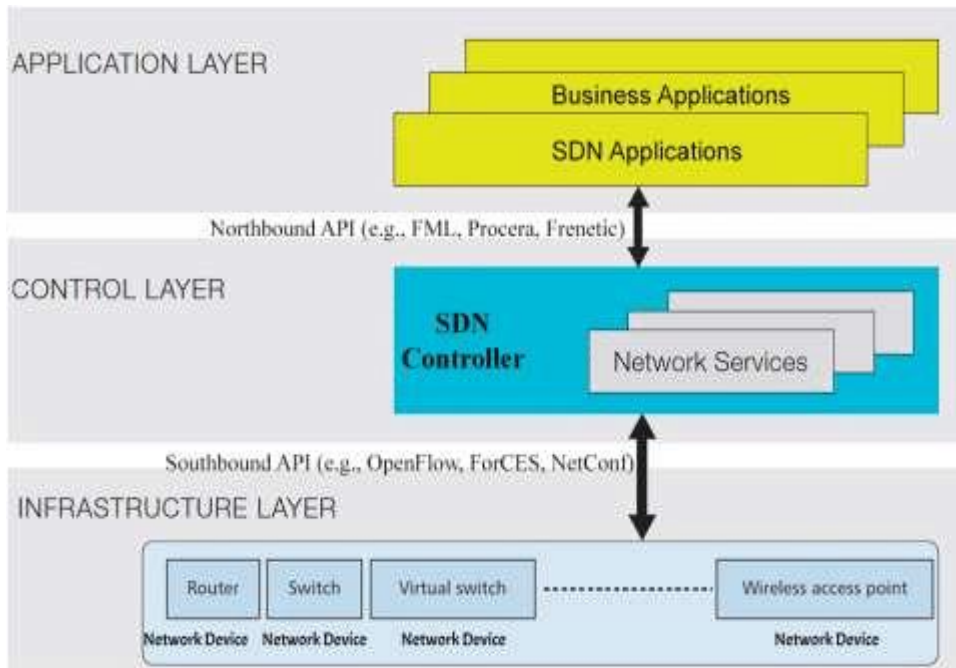


Figure 1: Software Defined Network Architecture

#### 4. OpenFlow

OpenFlow protocol, was implemented at Stanford University in 2008 and one year later, in 2009, Version 1.0 was released. The Open Networking Foundation (ONF) manages OpenFlow and defines it as the first standardized interface used for communication between Control and Data layer of a Software Defined Network. The next version of OpenFlow protocol released in February of 2011 by Open Networking Foundation (ONF). In 2012 OpenFlow version 1.2 was released. Current version of OpenFlow protocol is 1.5.1.

OpenFlow version 1.0 included only one flow table and 12 matching elements, as result, it had limitation and flexibility issues. In OpenFlow 1.1 version, these issues were prevented with the use of multiple flow tables, that allows the execution of more than one actions at the same time. OpenFlow 1.1 version, also introduced the use of a group table that includes four types of entries:

- All: enables functions that forward packets to multiple ports, such as multitasking.
- Select: enables functions that select one action bucket at a time, such as load balancing.
- Indirect: is used to divide flows into groups with same actions, it increases the function's efficiency.
- Fast failover: detects actions with active ports for execution.

Openflow version 1.2 released in order to get over the fixed length structure of OpenFlow 1.1 and the networking increasing needs. OpenFlow 1.2 new feature is the Type-Length-Value (TLV) structure that includes OpenFlow Extensive Match (OXM) which is a new, modular way to match fields using dynamic criteria. Furthermore, OpenFlow 1.2 supports IPv6 protocol and introduced the Controller role-change mechanism that allows the use of more than one controllers in order to implement more flexible networks, avoid limitations and network failures. There are three available Controller role in this mechanisms: master, slave and equal.

OpenFlow 1.3 released to provide the required quality of service to the network users. It introduces the Meter Table, that is a list of meter bands that define the OpenFlow switches behaviour. In addition, in OpenFlow 1.3, table-miss entry was added to the flow table and defines the behaviour of non-matched packets.

OpenFlow 1.4 introduces the Synchronized Table that is used to synchronize flow tables in two ways: bidirectionally and unidirectionally. In order to group multiple modifications in one group, OpenFlow 1.4 introduces Bundle, it is a feature that is created by the Controller and groups a set of modifications and applies them to multiple switches.

OpenFlow 1.5 extends Bundle by adding a new property, execution time that increases the synchronization between the switches. In addition,

OpenFlow 1.5 introduces the Egress Table that allows matching packet based on its port.

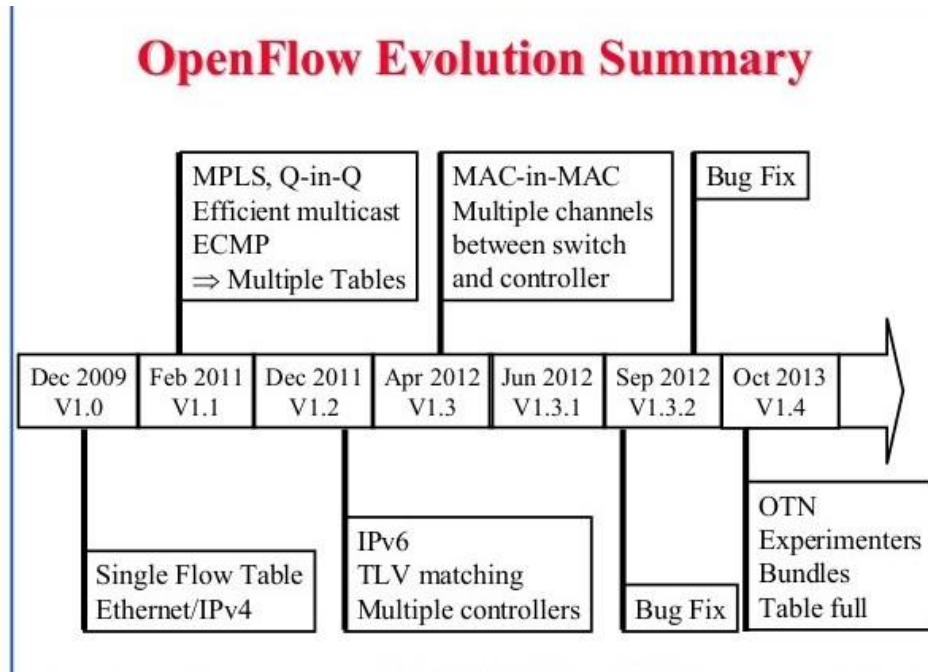


Figure 2: OpenFlow Evolution

The separation of Control level from network elements created the need of a communication protocol compatible with all network elements of all different vendors so that all of them could be managed by the SDN Controller. OpenFlow protocol is one of the first standardized SDN communication protocols that meet this requirement and at the same time it offers the needed security. The majority of network equipment vendors recognized the need to create network elements compatible with OpenFlow, these elements are known as OpenFlow switches. Each OpenFlow switch includes multiple tables that are used to manage the packet forwarding through the switch.



Flow Table matches the incoming packets to the flow and defines the functions that will be performed. An OpenFlow switch may contain more than one Flow Table with the following structure:

- Match Fields: Contain information about the port that the packet arrived (Ingress port), ethernet Source, destination address, TCP and UDP source and destination ports and IPv4 and IPv6 protocol numbers, source and destination addresses.
- Priority
- Counters: For example, incoming packets counter or transmitted bytes.
- Instructions: The set of actions to be performed if the packet matches the entry.
- Timeouts: Define the idle duration of the flow.
- Cookie

The latest OpenFlow versions also include the table-miss flow.

A flow could be defined as a set of forwarding packets that meet the same rules, stored in the Flow Tables of the OpenFlow switch. These rules are

known as actions or action sets and describe packet forwarding and group table processing operations. There are six different types of actions:

- Output: Packet forwarding.
- Set-queue: Sets queue ID for the forwarded packets to a specified port.
- Group: Groups packet process.
- Push/Pop-tag: Push or Pop a tag field.
- Set-Field: They set the Header Fields in the packet.
- Change TTL (Time To Live)

There are also four types of action sets:

- Direct Packet Through Pipeline: Directs the packets to the meter Table.
- Perform action on packet: Performs actions to the matched packets.
- Update action set: Modifies or clears the actions of the action set for a specified flow.
- Update metadata: Metadata carry information to the other tables.

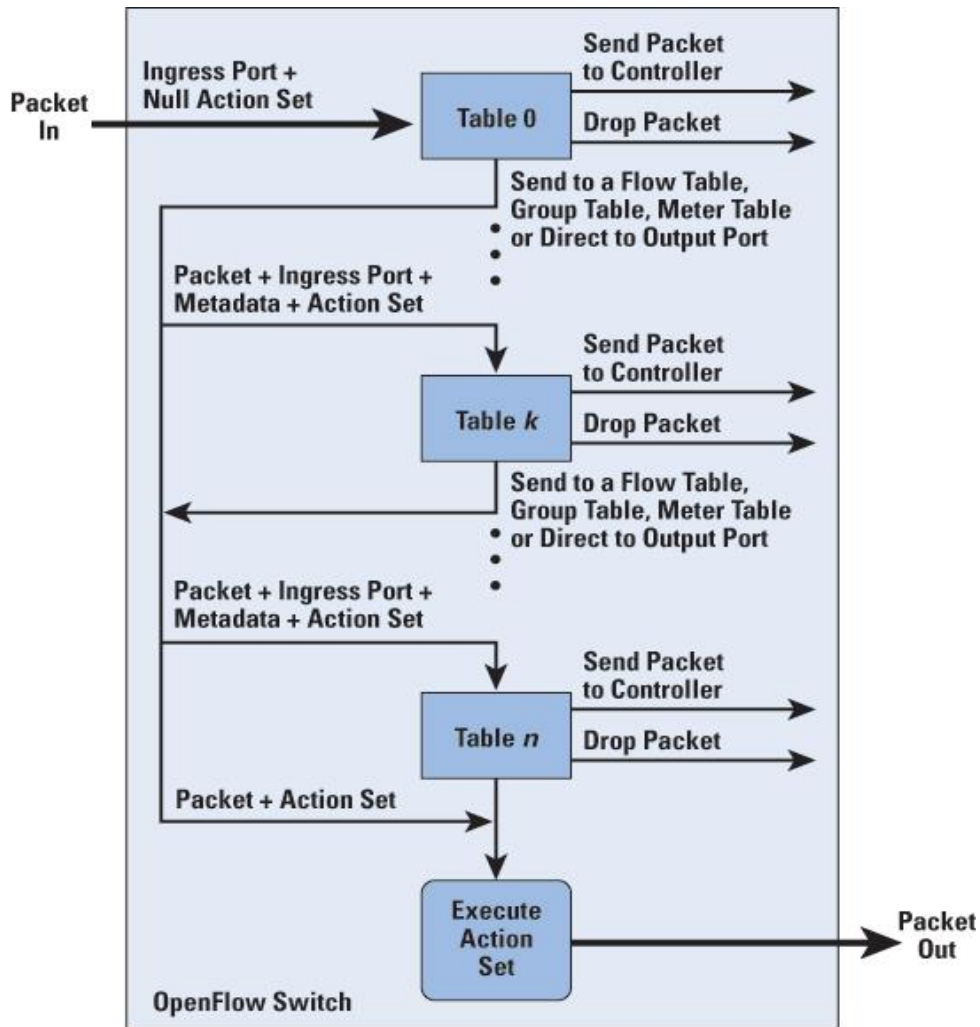


Figure 3: OpenFlow Switch Architecture

SDN Controllers use OpenFlow protocol to communicate and exchange messages with the switches. Openflow protocol is running over Transport Layer Security (TLS) or Secure Sockets Layer (SSL) in order to achieve the required network security and support three different kinds of messages:

- Controller to Switch: Messages sent from the controller in order to configure the flow and group Tables of the switch or forward a packet to the output port.

- Asynchronous: Messages sent from a switch to the Controller. For example, a packet that did not match to the flow table.
- Symmetric: Messages sent by Controller and switches when connection is established in order to verify the bandwidth, latency and the connection establishment.

Through these types of messages, the Controller manages the elements of complex networks and gets information about network performance in order control traffic forwarding and secure the Quality of Service requirements.



## 5. Analytics

Analytics is the examination of a data set using mathematics, statistics and other predictive and calculation techniques in order to draw conclusions about the recorded data. Businesses use analytics to make predictions, organize their business plans, achieve their goals and improve the efficiency. In addition, it is an important tool for every business that uses a large amount of data as they can determine risks, predict possible dangers and develop new, efficient solutions in order to avoid them. There are three different types of analytics:

- **Descriptive statistics:** They are models that provides information about what happened and the reason it happened. For example, the number of produced units of a product and how many are sold.
- **Predictive analytics:** Models that use past data in order to predict behaviours. They became more efficient with the increase of computing powers and development of new predictive techniques.
- **Prescriptive analytics:** Model that finds the solution to a prediction by providing information and efficient decisions. It demands lots of computing power and also the use of big data.

As shown above, analytics include more than collecting data. It is a complete process that collects data, uses data models to make accurate predictions and provide efficient solutions in order to make correct decisions.

### **5.1. Software Defined Network Analytics**

Software Defined Networking main characteristic is the separation between Control and Data layer. As a result, Software Defined Networks are more adaptable, flexible and programmable. SDN Analytics offer the required intelligence and visibility to the network.

Software Defined Networks may be complex because they often include different types of services such as Cloud services and voice over IP and also multiple applications with different network requirements running on them. It is easy to understand that optimization and management of dynamic networks would be almost impossible without automated control. In order to achieve the needed control automation, SDN architecture needs the provided intelligence from SDN analytics to recognize every service and application needs, avoid possible collisions and optimize network performance in real time. SDN analytics is an important network tool that applies numerical and visualization techniques on the available operational data to offer instant visibility of the network to administrators, help them manage potential issues and make the right decisions.

We conclude that network architectures could be based on SDN control automation and intelligence offered by analytics combined with multiple services and applications such as Cloud services and big data to implement flexible, dynamic, programmable and self-managing networks that would be able to control, configure and optimize their own elements automatically. Implementation of such networks could reduce operational costs increase productivity and at the same time offer increased quality of service compared to classic networking architectures.

---

# Software Defined Network Analytics Application

## **6. Application Environment setup and Development**

### **6.1. Application Development Environment**

SDN Analytics application was implemented in a Linux based virtual machine using IntelliJ IDEA. It is a Java integrated development environment, that includes many development features which make it a complete, fully-featured development environment. In addition, Apache Tomcat 8 web server is used and configured in IntelliJ. Client side of the application is developed in HTML, CSS and JavaScript and server side is developed in Java.

### **6.2. Environment setup**

#### **6.2.1. Mininet**

In order to set up an environment to develop and test the application, a virtual network is created using Mininet.

Mininet is a virtual network emulator that creates hosts, switches and links on a single Linux Kernel. Switches created in Mininet support OpenFlow and hosts run Linux software and behave like real machines. In addition, Mininet:

- Needs only few seconds to start up a network.
- Provides a simple network testbed for OpenFlow applications development as custom topologies can be easily created.
- Allows multiple developers to work on the same topology and run real applications, compatible to Linux.
- Supports repeatable system-level regression tests.
- Provides a topology and OpenFlow aware CLI for running and debugging tests.

- Provides an extensible Python API for network creation but on the other hand, it can be used without programming.
- Is an open source project.

Any application developed and tested on Mininet can be deployed on a real system without important changes as Mininet networks run real code and standard Linux applications. OpenFlow switches and hosts communicate over the network using virtual ethernet, hosts can also send packets with a given speed and delay and their behaviour is similar to hardware network equipment.

On the other hand, Mininet has some limitations:

- It is unable to extend the available CPU or bandwidth on a single server.
- It is incompatible to OpenFlow switches and applications that do not run Linux.

### 6.2.2. OpenDaylight Controller

OpenDaylight Controller is a multi-protocol scalable controller developed for SDN deployments. It is an open project with the ability to deploy multiple network environments. Its main advantage is the support of OpenFlow protocol and all open Software Defined Networking standards.



In April 2013, OpenDaylight announced by Linux Foundation cooperation with some high profile companies such as Cisco, Juniper, Microsoft, VMWare and HP. Hydrogen was OpenDaylight first release and launched in February 2014 and consists of an open code controller with some protocol plug-ins and virtualization capabilities. Eight months later, in November 2014 the second release, named Helium, was launched. It provided a Karaf based work environment and model driven network management. In June 2015, Lithium release was launched. It introduced new plug-ins to support new network protocols and a model driven management based OpenFlow plug-in that supports OpenFlow versions 1.0 to 1.3. OpenDaylight Beryllium released in February 2016 and included performance improvements, also improved YANG tools, NETCONF and RESTCONF protocol features. The

most recent release of OpenDaylight is Boron, it launched in November 2016 and includes new development and operational tools.



Figure 3: OpenDaylight Controller Evolution

OpenDaylight Controller includes a service abstraction layer (SAL) and provides model driven service abstraction. SAL separates northbound services and application plugins from southbound protocol plugins and its main purpose is to adapt southbound functions to application and service functions that are generated by OpenDaylight Controller northbound API. Model driven SAL (MD-SAL) provides Northbound and Southbound interfaces a common REST API for application and feature development and stores data model defined by plug-ins. Supported Model Driven protocols are NETCONF and REST-CONF. OpenDaylight Controller also supports YANG modelling language. More extensively:

- NETCONF: It is a network management protocol supports Remote Procedure Calls(RPC) and configures data stores using

Create, Update and Delete (CRUD) operations. NETCONF uses XML for configuration and protocol messages.

- REST-CONF: Provides an interface over HTTP for accessing data and data stores defined respectively in YANG model and NETCONF protocol. The exposed resources are encoded in JSON or XML and are retrieved and modified with HTTP GET, PUT DELETE and POST methods.
- YANG: It is a modelling, data-oriented language that models complex data sets as tree structures and generate APIs for application development. YANG also models RPCs to suit in model driven systems as Interface Description Language (IDL).

---

#### NETCONF and YANG in Context

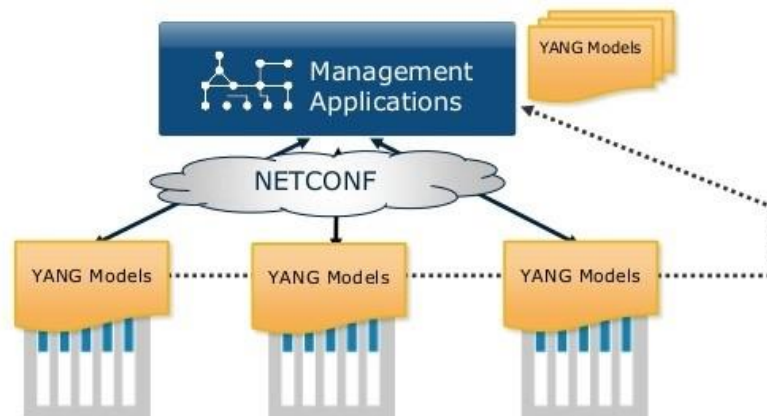


Figure 4: NETCONF & YANG model

In addition, OpenDaylight Controller is implemented in software and is contained in its own JVM (Java Virtual Machine). As a result, it can be deployed



on any operating system that supports Java. It also depends on the following technologies:

- MAVEN: It is a management tool that helps developers to manage the required plug-ins and provides automated dependencies.
- JAVA: It is the Object Oriented programming language that developers use to develop applications and features in the OpenDaylight Controller.
- OSGi (Open Service Gateway interface): It is the Controller's back end. OSGi binds modules for exchanging information and allows dynamic load of bundles.
- KARAF: It is an application container that makes applications installation simple.

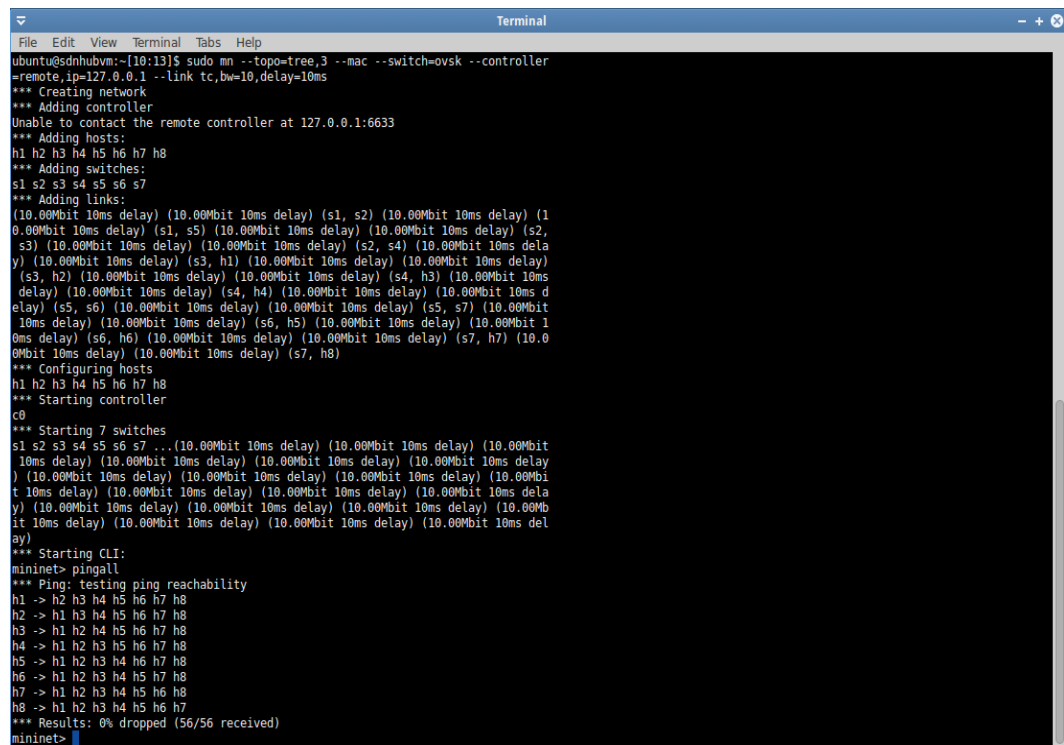
In this study, we will use OpenDaylight DLUX which is a user interface.

Application designed for OpenDaylight Controller. It includes multiple features that provide detailed information about network architecture and performance. OpenDaylight DLUX uses SAL services to get network related information.

## 6.3. SDN Analytics application

### 6.3.1. Application Environment Setup

First of all, we deployed a virtual network with Mininet emulator. The network includes seven OpenFlow switches connected each other with ethernet in a tree topology and eight connected hosts.



```
Terminal
File Edit View Terminal Tabs Help
ubuntu@sdnhubvm:~[10:13]$ sudo mn --topo=tree,3 --mac --switch=ovsk --controller
=remote_ip=127.0.0.1 --link tc,bw=10,delay=10ms
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s1, s2) (10.00Mbit 10ms delay) (1
0.00Mbit 10ms delay) (s1, s5) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s2,
s3) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s2, s4) (10.00Mbit 10ms dela
y) (10.00Mbit 10ms delay) (s3, h1) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay)
(s3, h2) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s4, h3) (10.00Mbit 10ms d
elay) (10.00Mbit 10ms delay) (s4, h4) (10.00Mbit 10ms delay) (10.00Mbit 10ms d
elay) (s5, s6) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s5, s7) (10.00Mbit
10ms delay) (10.00Mbit 10ms delay) (s6, h5) (10.00Mbit 10ms delay) (10.00Mbit 1
0ms delay) (s6, h6) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s7, h7) (10.0
0Mbit 10ms delay) (10.00Mbit 10ms delay) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ... (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms dela
y) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbi
t 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms dela
y) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mb
it 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms del
ay)
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```

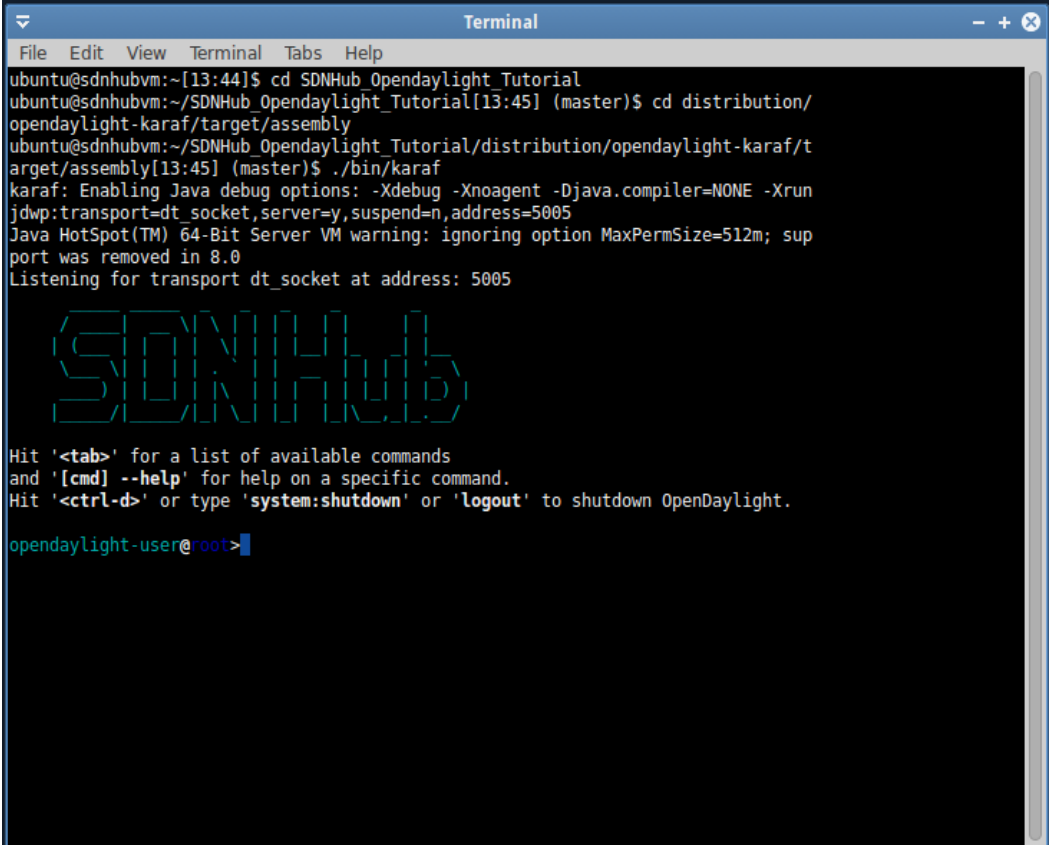
Figure 5: Mininet deployment through terminal

The network deployment command is shown in the first line of the terminal in Figure 5 and it consists of the following properties:

- `sudo mn`: Mininet console initialization.
- `--topo=tree`: Commands Mininet to deploy a tree topology.
- `--mac`: Allocates MAC addresses to the hosts according to their IP addresses.
- `--switch=ovsk`: Commands Mininet to use ovsk switches.
- `--link tc, bw=10, delay=10ms`: Adds 10 Mb per second Bandwidth and 10msec delay to the virtual network.

Once the network is deployed, a “pingall” command is executed in order to ensure that there is reachability between network hosts.

Secondly, OpenDaylight Controller was activated in order to manage and control the deployed network.



```
ubuntu@sdnhubvm:~[13:44]$ cd SDNHub_Opendaylight_Tutorial
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial[13:45] (master)$ cd distribution/
opendaylight-karaf/target/assembly
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial/distribution/opendaylight-karaf/t
arget/assembly[13:45] (master)$ ./bin/karaf
karaf: Enabling Java debug options: -Xdebug -Xnoagent -Djava.compiler=NONE -Xrun
jdwp:transport=dt_socket,server=y,suspend=n,address=5005
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; sup
port was removed in 8.0
Listening for transport dt_socket at address: 5005

SDNHub

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Figure 6: OpenDaylight Controller deployment through terminal

Then we enter the OpenDaylight Dlux web user interface through the following Url: <http://localhost:8181/index.html>. Username and password are both “admin” by default.

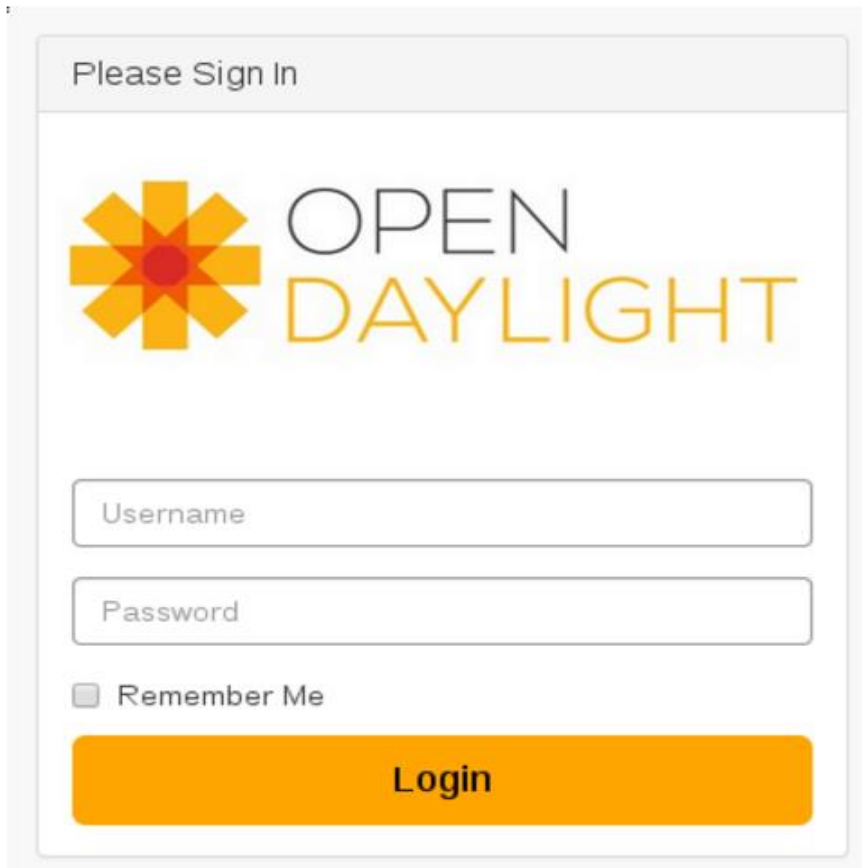


Figure 7: OpenDaylight Dlux login page

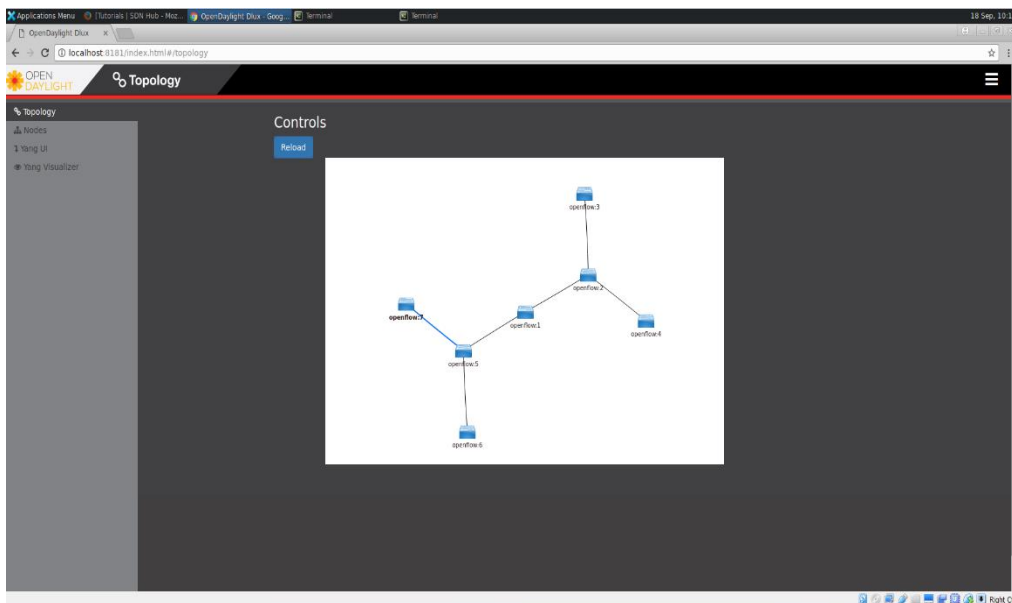


Figure 8: OpenDaylight Dlux main page

OpenDaylight Dlux provides visibility of the network topology and information about network statistics as well. But one of the most important features of OpenDaylight Dlux is the Yang UI, the third choice of the menu at the left of the main screen (Figure 8).

Yang UI provides all necessary APIs and subAPIs for development to the application layer. On the top part of Yang UI page are displayed all APIs and subAPIs in tree formation and all the available functions that could be used when an API is selected (GET, PUT, POST, DELETE, etc). The required path for every operation is also displayed under the API tree formation, before the function buttons. On the bottom part of Yang UI page is displayed the list of elements of the selected subAPI.

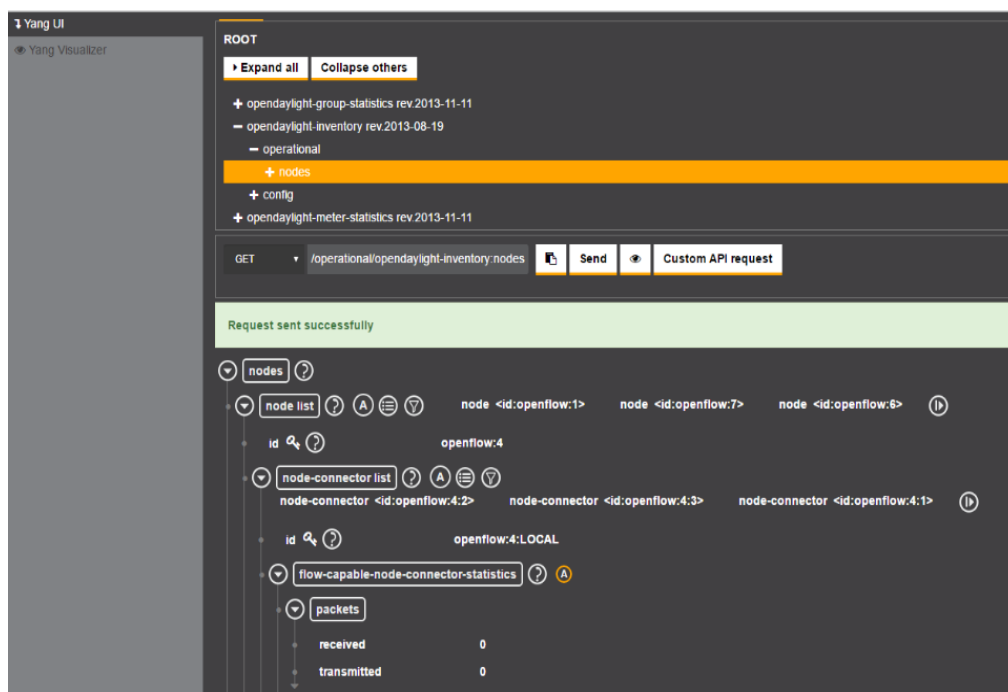


Figure 9: Yang UI page

Finally, we used the generated API from Yang UI to develop a Network Analytics application that presents graphically the network performance and using network statistics, provided by Yang UI.

### **6.3.2. Application Development**

The Network Analytics application is implemented using Jersey Framework which is an open source framework for developing RESTful Web Services in Java. The generated statistics are in Json format and we used Google Charts for the graphical presentation. When each page of the application is loaded, for each chart, a REST call is sent to the back-end part of the JAVA application. Then, A REST call from the application to the controller modifies the statistics as they come in Json format and makes them available for presentation from Google charts. Opendaylight Controller provides the application with a list of the nodes and their statistics for each chart. In order to create a list with the Node IDs and the needed values, we use a Map in Java which is filled with all the needed values with the use of a for loop. Map is a collection in Java that pairs a value for a key as `HashMap<value, key>` and we used a `HashMap` for every chart. The application is deployed using Apache Tomcat 8 server. It consists of three tabs.

The first tab presents two column charts and three pie charts. The first column chart shows the transmitted bytes, drops and errors of the whole network depending on time. The second column chart shows the received bytes, drops and errors of the whole network depending on time. The first pie chart shows the duration percentage of an operation of each flow connector, the second pie chart presents the percentage of transmit drops of each flow connector and finally, the third pie chart presents the crc-Errors respectively. If the transmit drops and crc-Errors pie charts does not have any values, then a message is printed instead of the charts. CRC (Cyclic Redundancy Check) Errors could be caused by a damaged transmitted packet and the receiver demands packet retransmission.

The second tab, includes more specific charts. There are five column charts and a pie chart that presents the statistics of every flow connector. The first two charts present the transmitted and received bytes of every flow connector and the third chart shows the duration of a traffic operation on every flow connector. On the second row, the first two charts present the transmitted and received packets for every flow connector and the last pie chart shows the transmitted errors for every flow-connector of the virtual network.

Finally, the third tab is created to present a summary of the two other tabs and show the most productive and efficient nodes. On the other hand, in this page are presented also the most defective flows of the network and them with the lowest performance. In this page, there are two column charts, two bar charts and two pie charts. More specific, the first chart of this page is a column chart that presents the five flow connectors with the biggest amount of transmitted bytes. Secondly, there is a pie chart that presents the five flow connectors with the most network collisions. The last chart of the first row is a bar chart that presents the five flow connectors with the biggest amount of transmitted packets. In the second row, the first column chart presents the five flow connectors with the most received bytes. The next chart is a pie chart that presents the five flow connectors with the most transmit drops and the last one is a bar chart that shows the five flow connectors with the biggest amount of received packets.

### **6.3.3. Application Scenario**

Once the application environment has been set up and the application is deployed successfully, three different types of network topologies will be created and compared. First of all, a linear topology with five switches and two hosts per switch will be deployed. The second one is a ring topology with five switches and two hosts per switch. The third one is a tree topology with seven switches and eight hosts.

The next step is to examine how these network topologies react with different performance parameters such as different bandwidth, increased delay and loss. Firstly, the network topologies will be compared without any performance parameters. Then, two different sets of performance parameters will be used for each network topology.

The first performance parameter set includes 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss. The second performance parameter set includes 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

### **6.3.3.1 Linear network Topology**

Linear topology is a kind of network topology where all the switches are connected one after the other like a chain using two ways links. The main disadvantage of this kind of network topology is that if a link of the network is severed, then the whole network transmission is halted. On the other side, linear network topologies are ideal for small networks as they are easy to set up and they request smaller amount of cables. Although, in larger networks where more devises are connected the speed could be reduced.

In this study, we are going to deploy a custom linear network topology, using the following python script in mininet,



```

from mininet.node import CPULimitedHost
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.link import TCLink

class SimplePktSwitch(Topo):

def __init__(self, **opts):

    super(SimplePktSwitch, self).__init__(**opts)

    # Add hosts and switches
    h1 = self.addHost('h1')
    h2 = self.addHost('h2')
    h3 = self.addHost('h3')
    h4 = self.addHost('h4')
    h5 = self.addHost('h5')
    h6 = self.addHost('h6')
    h7 = self.addHost('h7')
    h8 = self.addHost('h8')
    h9 = self.addHost('h9')
    h10 = self.addHost('h10')

    # Adding switches
    s1 = self.addSwitch('s1', dpid="0000000000000001")
    s2 = self.addSwitch('s2', dpid="0000000000000002")
    s3 = self.addSwitch('s3', dpid="0000000000000003")
    s4 = self.addSwitch('s4', dpid="0000000000000004")
    s5 = self.addSwitch('s5', dpid="0000000000000005")

    # Add links
    self.addLink(h1, s1, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h2, s2, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h3, s3, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h4, s4, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h5, s5, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h6, s1, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h7, s2, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h8, s3, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h9, s4, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(h10, s5, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)

    self.addLink(s1, s2, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(s2, s3, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(s3, s4, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
    self.addLink(s4, s5, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)

def run():
    c = RemoteController('c', '0.0.0.0', 6633)
    net = Mininet(topo=SimplePktSwitch(), host=CPULimitedHost, controller=None, link=TCLink)
    net.addController(c)
    net.start()

    CLI(net)
    net.stop()

# if the script is run directly (sudo custom/optical.py):
if __name__ == '__main__':
    setLogLevel('info')
    run()

```

That includes five switches and two hosts per switch, ten hosts in total as shown in the next pictures:

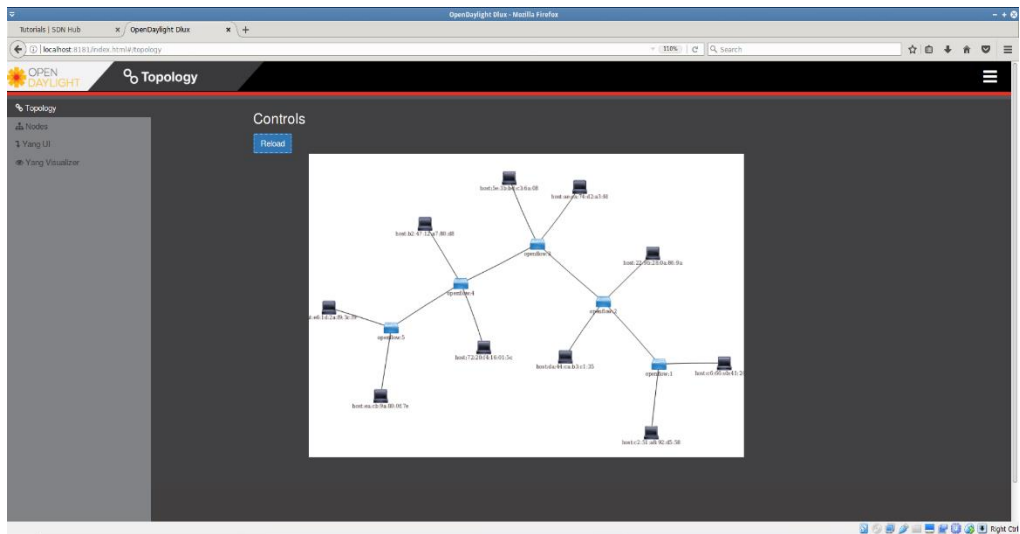
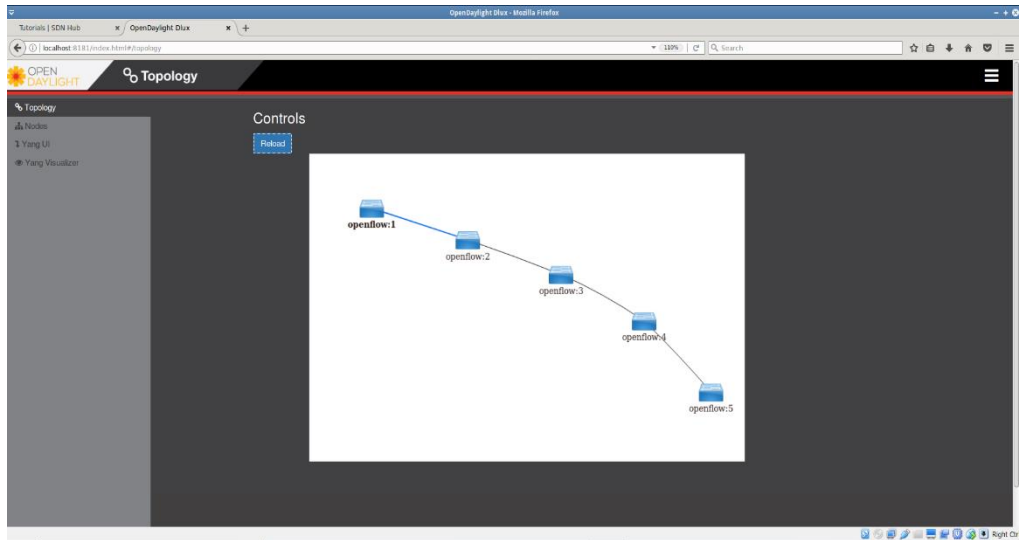


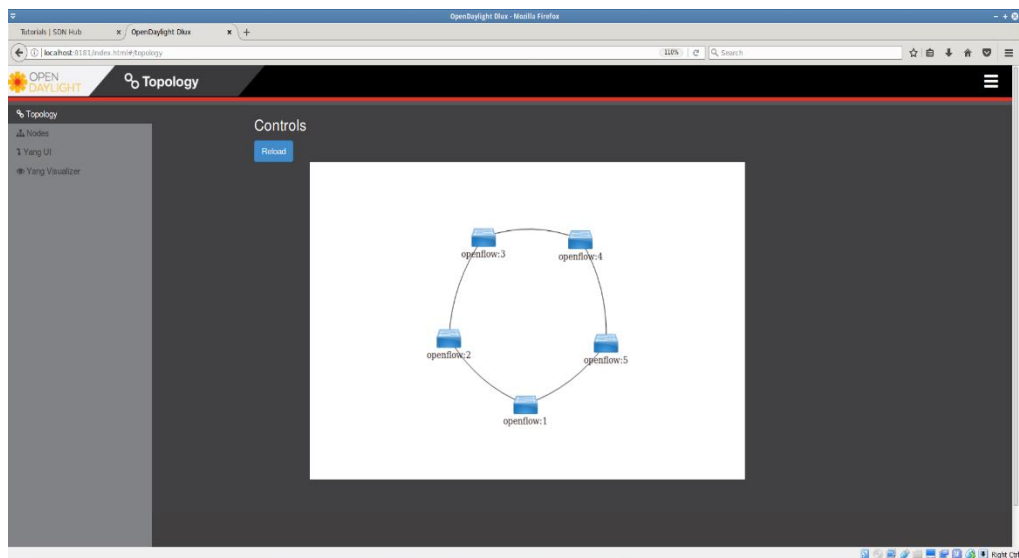
Figure 10, 11: Linear network topology with five switches and ten hosts.

### 6.3.3.2 Ring network Topology

In Ring network topologies, switch connections create a circular path. Each network element is connected with to others. There are two different kinds of ring network topologies. The unidirectional ring network topology which allows the packets to move in one direction, and the bidirectional ring network topology, where packets move in either directions.

Ring network topologies were used for smaller networks, however, today they are used in larger local or wide area networks as they offer increased performance, they are simple and easy to support. Ring network topologies offer decreased chance of network collisions as all packets move in one direction and packets can be transferred in high speed as well. Finally, hosts can be easily added or removed without affecting the network performance. On the other hand, star network topology has the same disadvantage as linear topologies, if a link of the network faces any issue, then the whole network will be affected. In addition, all data passes through each switch, this fact may affect the network performance.

The Ring network topology that is going to be used in our study includes five switches and ten hosts (two hosts per switch).



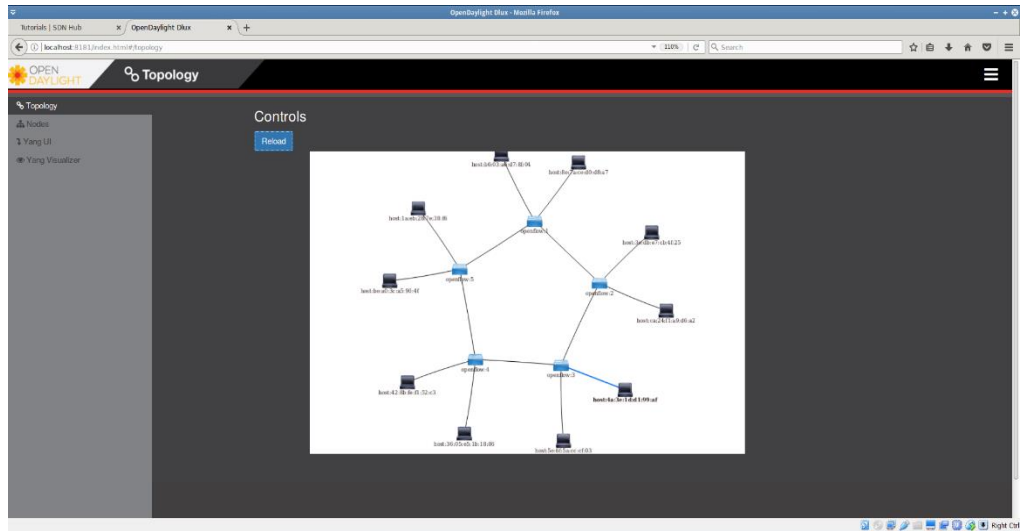


Figure 12, 13: Ring network topology with five switches and ten hosts.

The topology is created by the following python script:

```

from mininet.node import CPULimitedHost
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.link import TCLink

class SimplePktSwitch(Topo):
    def __init__(self, **opts):
        # Initialize topology
        # It uses the constructor for the Topo class
        super(SimplePktSwitch, self).__init__(**opts)

        # Add hosts and switches
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')

        h6 = self.addHost('h6')
        h7 = self.addHost('h7')
        h8 = self.addHost('h8')
        h9 = self.addHost('h9')
        h10 = self.addHost('h10')

        # Adding switches
        s1 = self.addSwitch('s1', dpid="0000000000000001")
        s2 = self.addSwitch('s2', dpid="0000000000000002")
        s3 = self.addSwitch('s3', dpid="0000000000000003")
        s4 = self.addSwitch('s4', dpid="0000000000000004")
        s5 = self.addSwitch('s5', dpid="0000000000000005")

        # Add links
        self.addLink(h1, s1, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h2, s2, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h3, s3, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h4, s4, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h5, s5, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)

        self.addLink(h6, s1, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h7, s2, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h8, s3, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h9, s4, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(h10, s5, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)

        self.addLink(s1, s2, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(s2, s3, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(s3, s4, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(s4, s5, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)
        self.addLink(s5, s1, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True)

    def run():
        c = RemoteController('c', '0.0.0.0', 6633)
        net = Mininet(topo=SimplePktSwitch(), host=CPULimitedHost, controller=None, link=TCLink)
        net.addController(c)
        net.start()

        CLI(net)
        net.stop()

# if the script is run directly (sudo custom/optical.py):
if __name__ == '__main__':
    setLogLevel('info')
    run()

```

### 6.3.3.3 Tree network Topology

Tree topology is a structure where network elements are connected like branches of a tree. It is also known as star bus network topology as it is a combination of a star and a bus topology's characteristics. More specifically, A number of star subnetworks are connected using a Bus, which is a common connection cable. The advantages of Tree topology are:

- Scalability that standalone star and bus topologies are unable to offer.
- Network can be easily expanded.
- Network can be divided to more star networks. It offers easier control and network maintenance.
- Easy error detection and correction.
- Tree “branches” are independent. For example, if a segment is damaged, the others are not affected.

On the other hand, there are several disadvantages on the use of Tree topology:

- The whole network depends on the Bus cable.
- Maintenance becomes more complicated when more star topologies are added.

The tree network topology that will be used for this research includes seven OpenFlow switches and eight hosts. (Figure 10)

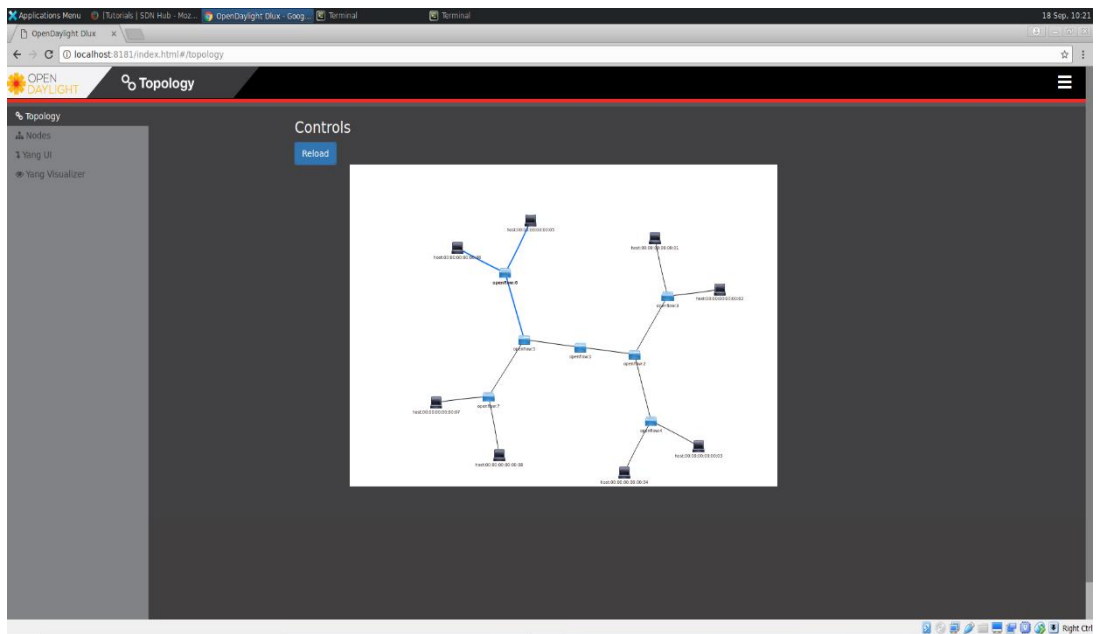
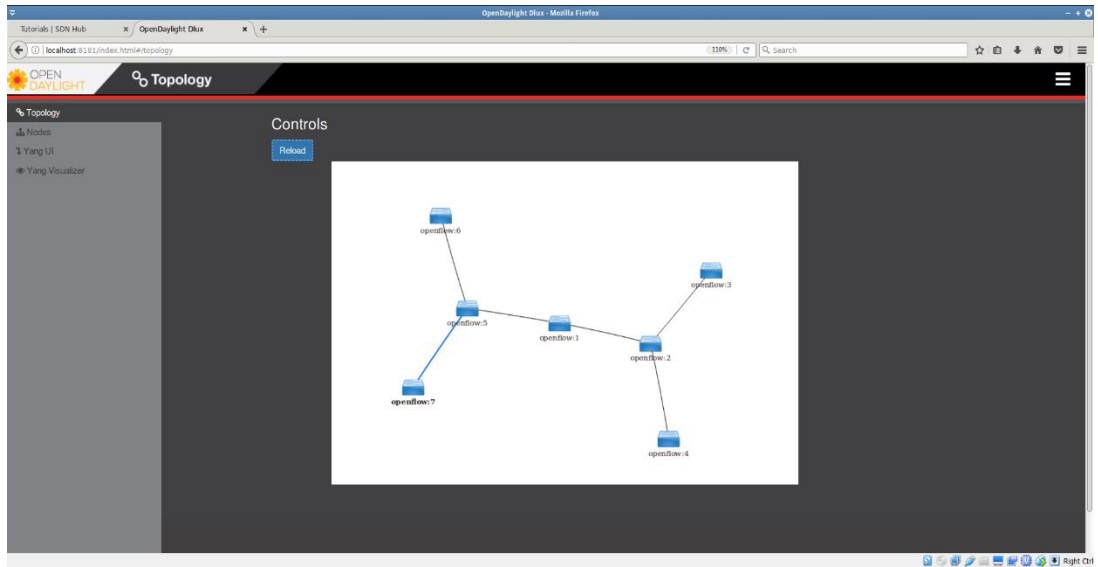


Figure 14, 15: Tree topology with five switches and ten hosts.

The topology is created using the following python script:

```

from mininet.node import CPULimitedHost
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.link import TCLink

class GenericTree(Topo):

    def build( self, depth=1, fanout=2 ):
        # Numbering: h1..N, s1..M
        self.hostNum = 1
        self.switchNum = 1

    def build( self, depth=1, fanout=2 ):
        # Numbering: h1..N, s1..M
        self.hostNum = 1
        self.switchNum = 1
        # Build topology
        self.addTree(depth, fanout)

    def addTree( self, depth, fanout ):

        isSwitch = depth > 0
        if isSwitch:
            node = self.addSwitch( 's%s' % self.switchNum )
            self.switchNum += 1
            for _ in range( fanout ):
                child = self.addTree( depth - 1, fanout )
                self.addLink( node, child, bw=7, delay='15ms', loss=30, max_queue_size=5000, use_htb=True )
        else:
            node = self.addHost( 'h%s' % self.hostNum )
            self.hostNum += 1
            return node

def run():
    c = RemoteController('c', '0.0.0.0', 6633)
    # Change the args of GenericTree() to your desired values. You could even get them from command line.
    net = Mininet(topo=GenericTree(depth=3, fanout=2), host=CPULimitedHost, controller=None, link=TCLink)
    net.addController(c)
    net.start()

    # installStaticFlows( net )
    CLI(net)
    net.stop()

# if the script is run directly (sudo custom/optical.py):
if __name__ == '__main__':
    setLogLevel('info')
    run()

```



### 6.3.4. Results Analysis

The SDN Analytics Application will be presented in two sections. Firstly, we are going to compare each network topology for different performance parameters. Two performance parameter sets were used for each topology and it will be presented without the use of performance parameters as well.

Secondly, all the topologies will be compared according to the performance parameters that are used.

#### 6.3.4.1. Linear Topology Analysis

In the following pictures, the first page of the SDN Analytics application is presented for the linear network topology. The first picture presents the total network's performance statistics without any performance parameters, In the second picture, there were added several performance parameters as we described before. They are 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss. In the third picture, the parameters are 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

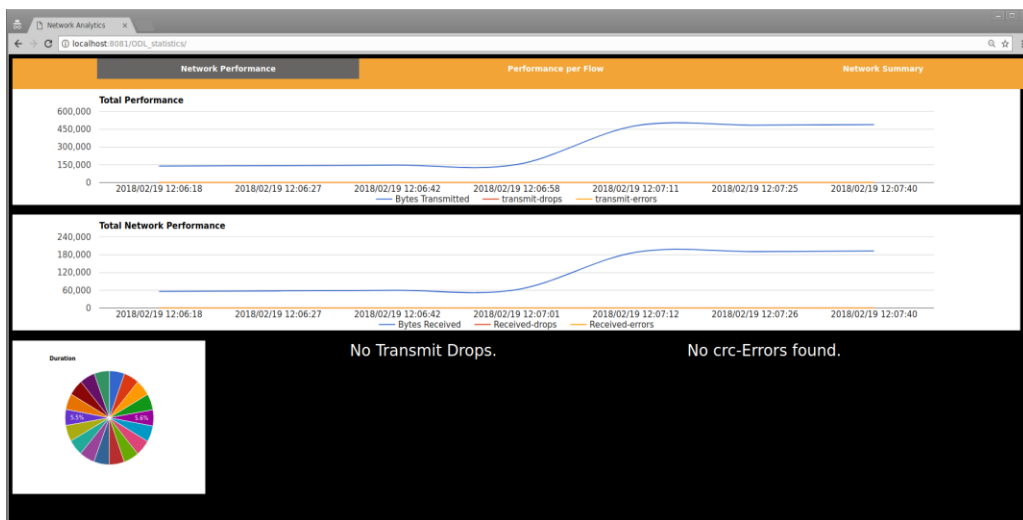


Figure 16: SDN Analytics application first page for linear network topology without performance parameters.

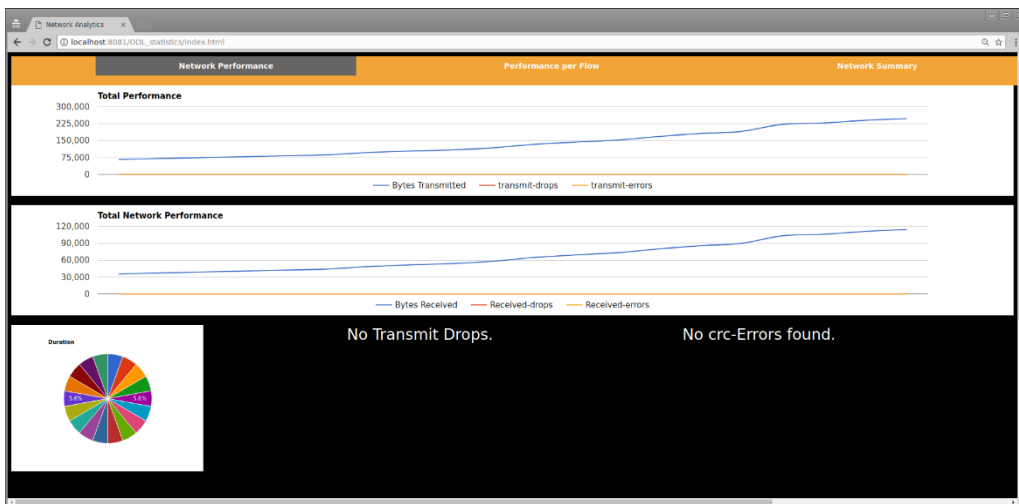


Figure 17: SDN Analytics application first page for linear network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

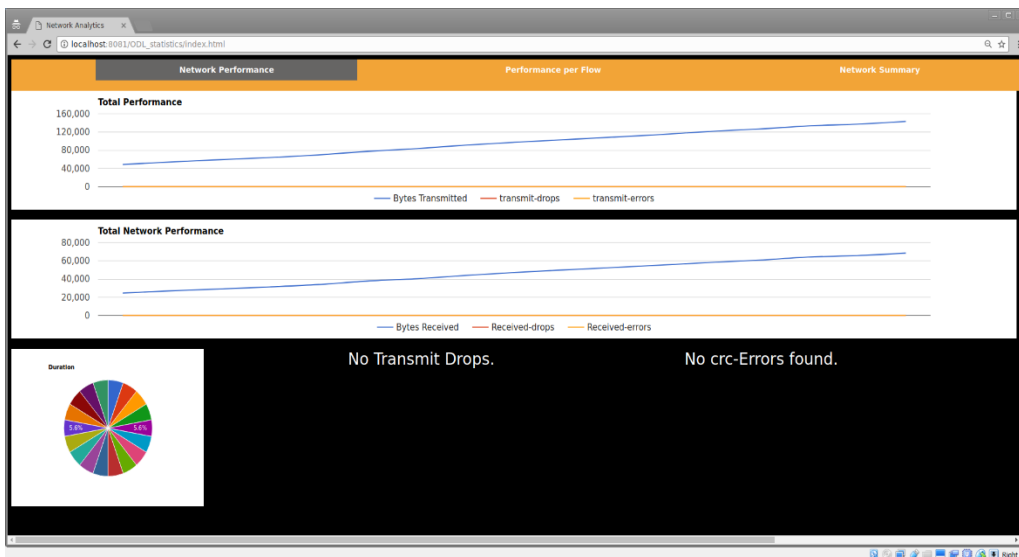


Figure 18: SDN Analytics application first page for linear network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss

The main difference between these network topologies with different performance parameters is that when traffic is generated, we are able to see the instant increase of transmitted and received bytes for the first picture. In the other two cases, the delay that was added is obvious and the amount of

transmitted and received bytes is decreased as well for the same transmission duration as it is shown in the duration pie chart.

The SDN Analytics application second pages for the linear topologies with different performance parameters are presented below:

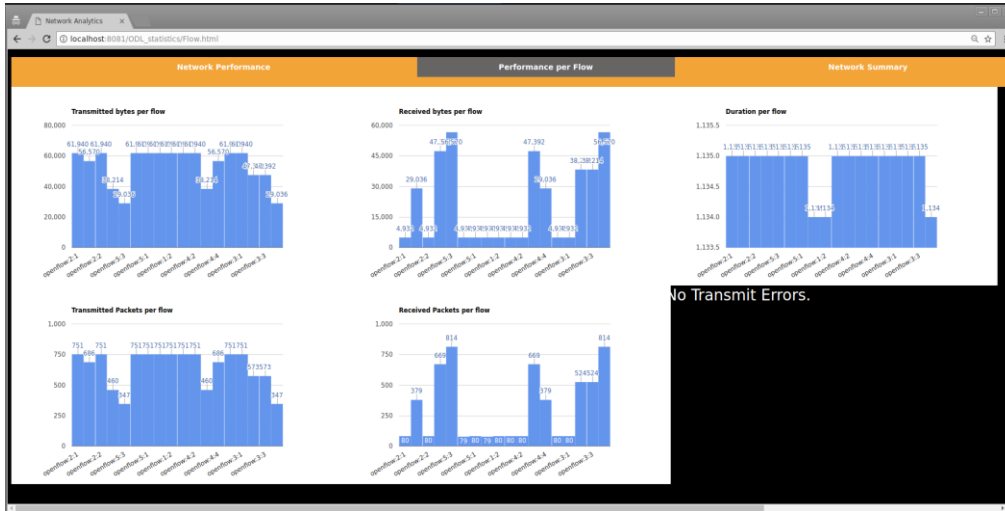


Figure 19: SDN Analytics application second page for linear network topology without performance parameters.

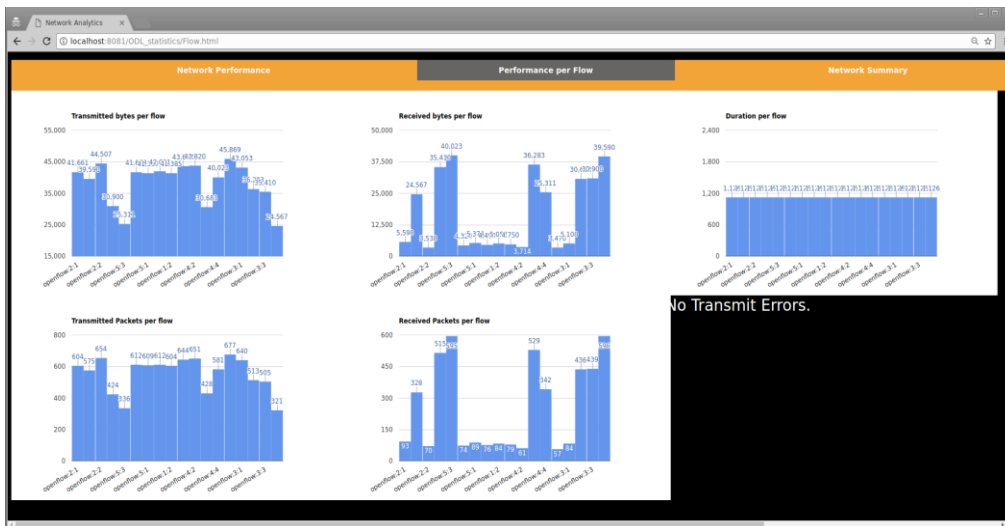


Figure 20: SDN Analytics application second page for linear network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

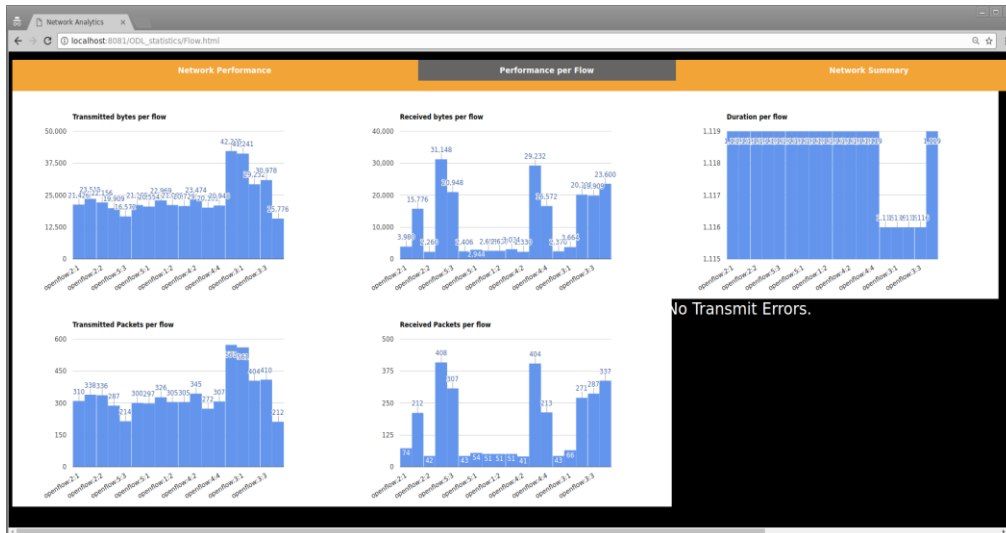


Figure 21: SDN Analytics application second page for linear network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss

As we expected, the main difference between these pages is that the amount of transmitted and received bytes and packets is decreased for the performance parameters with bigger values. It is obvious in every chart in these pages if we consider that the duration is almost the same for all of these cases.

The SDN Analytics application third pages for the linear topologies with different performance parameters are presented below:

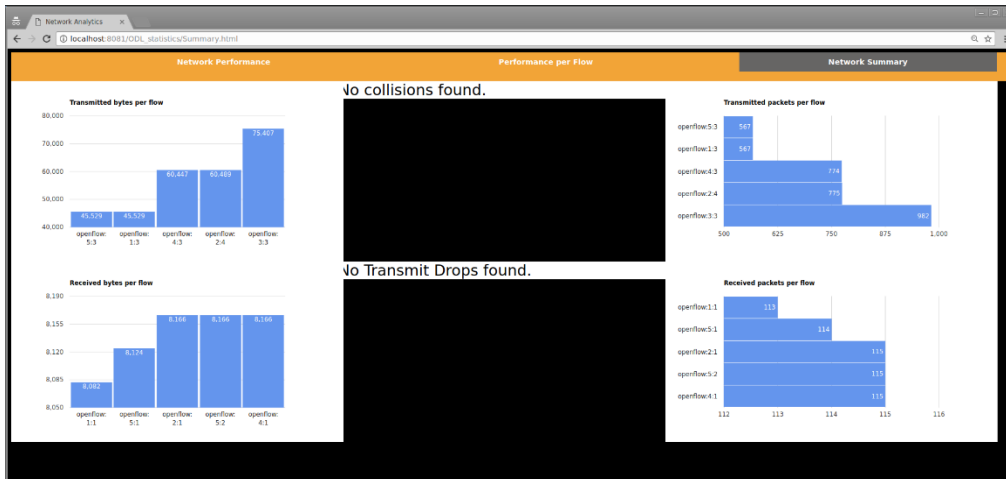


Figure 22: SDN Analytics application third page for linear network topology without performance parameters.

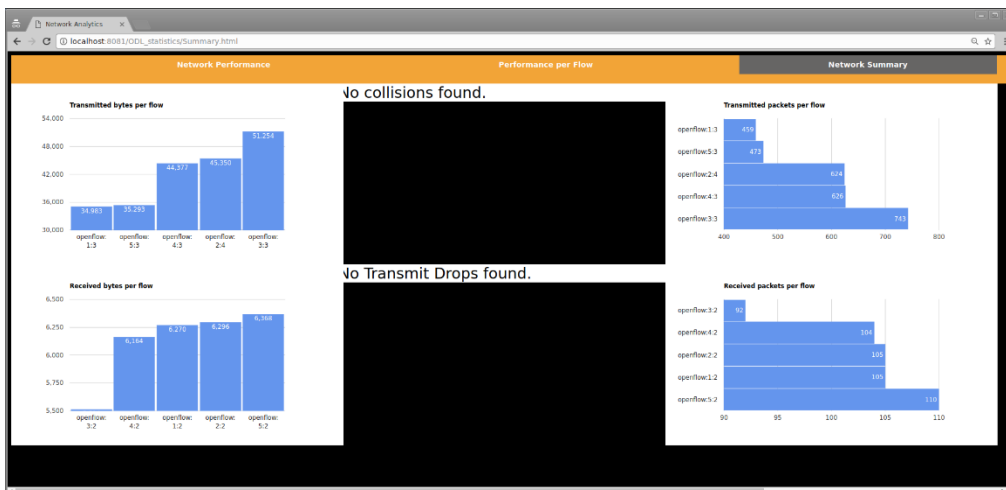


Figure 23: SDN Analytics application third page for linear network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

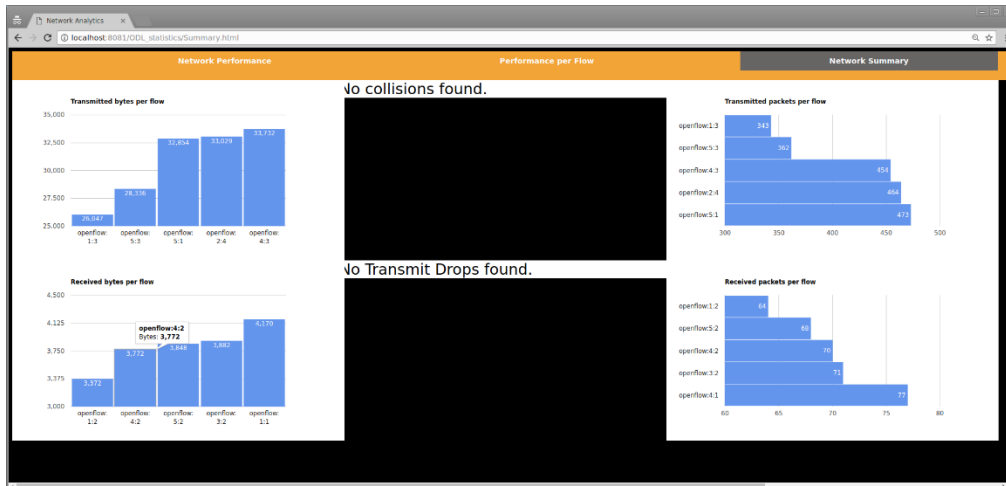


Figure 24: SDN Analytics application third page for linear network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

The third page of the SDN Analytics application is the summary page of the application. The amount of bytes and packets of the five most productive flow connectors of the topology without performance parameters is almost 50% bigger than the amount of bytes and packets of the five most productive flow connectors of the topology with the second set of performance parameters. The amount of bytes and packets of the topology with the first set of performance parameters is almost 25% reduced comparing it to the topology without performance parameters. On the other hand, there are no collisions or drops for any linear topology used in our study.

### 6.3.4.2 Ring Topology Analysis

In the following pictures, the first page of the SDN Analytics application is presented for the Ring network topology. The first picture presents the total network's performance statistics without any performance parameters, In the second picture, there were added several performance parameters as we described before. They are 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss. In the third picture, the parameters

are 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

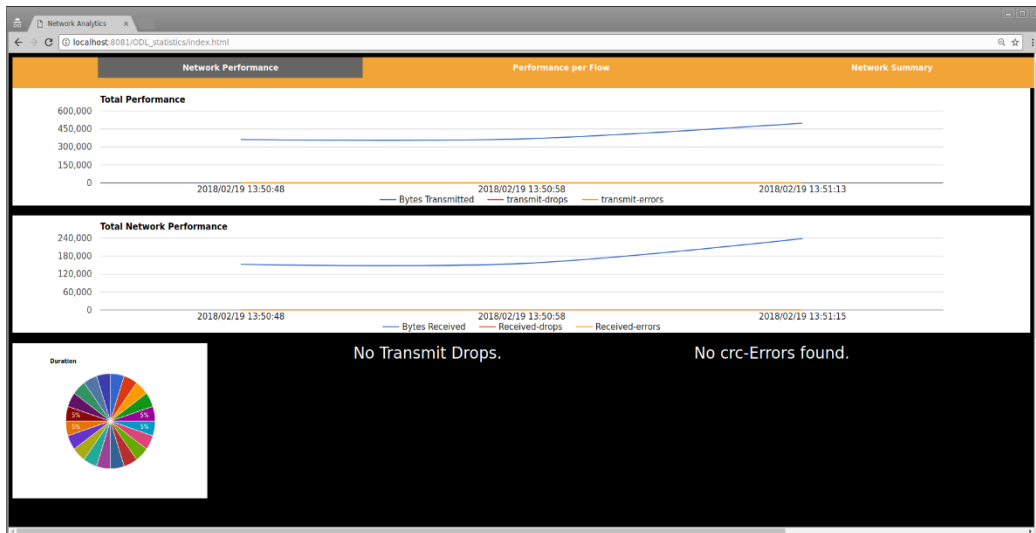


Figure 25: SDN Analytics application first page for Ring network topology without performance parameters.

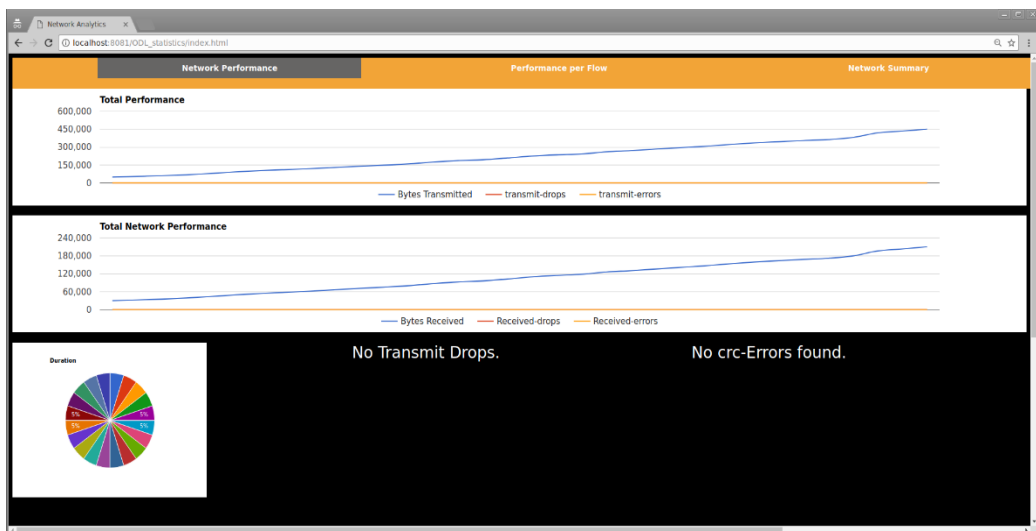


Figure 26: SDN Analytics application first page for Ring network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

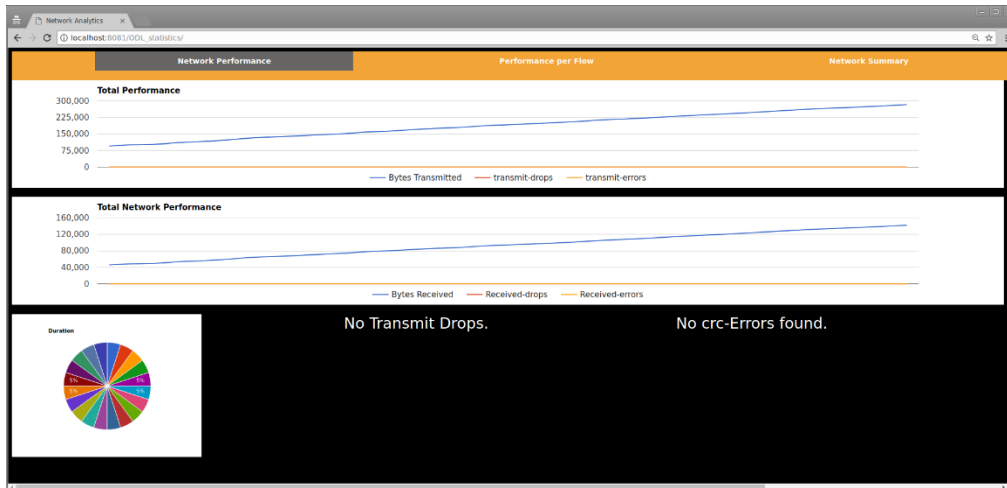


Figure 27: SDN Analytics application first page for Ring network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

The main difference between these network topologies with different performance parameters is that when traffic is generated, we are able to see the instant increase of transmitted and received bytes for the first picture. In the other two cases, the delay that was added is obvious and the amount of transmitted and received bytes is decreased as well for the same transmission duration as it is shown in the duration pie chart.

The SDN Analytics application second pages for the Ring topologies with different performance parameters are presented below:



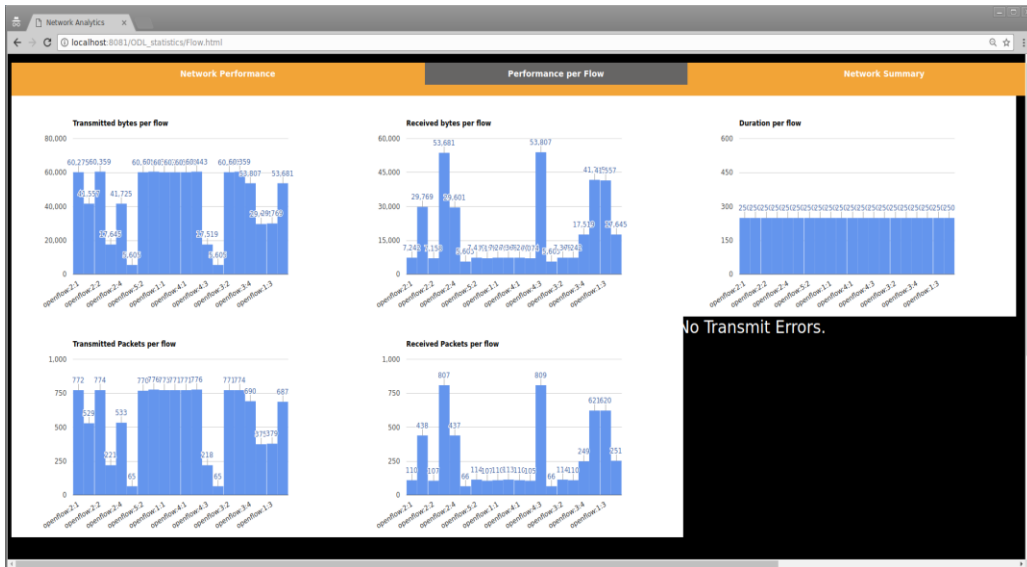


Figure 28: SDN Analytics application second page for Ring network topology without performance parameters.

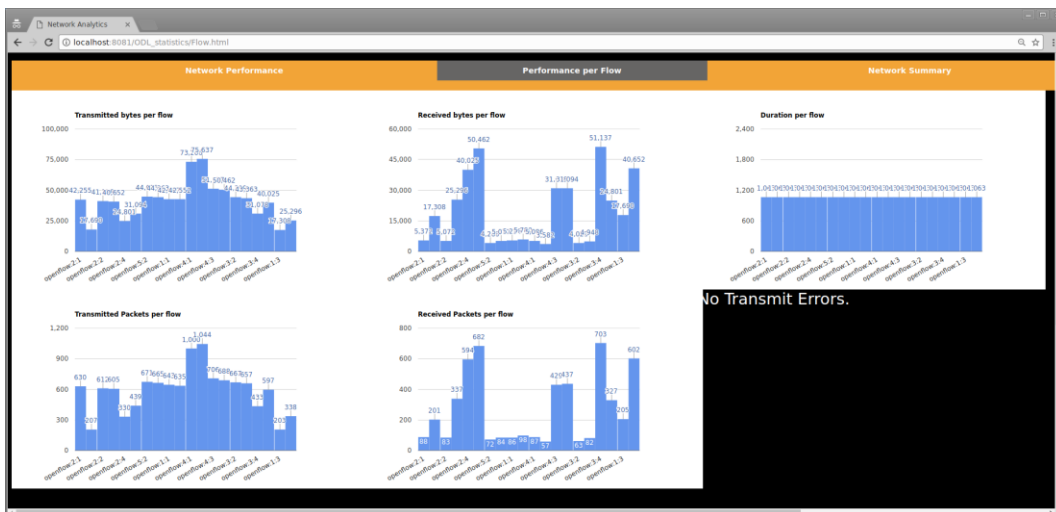


Figure 29: SDN Analytics application second page for Ring network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

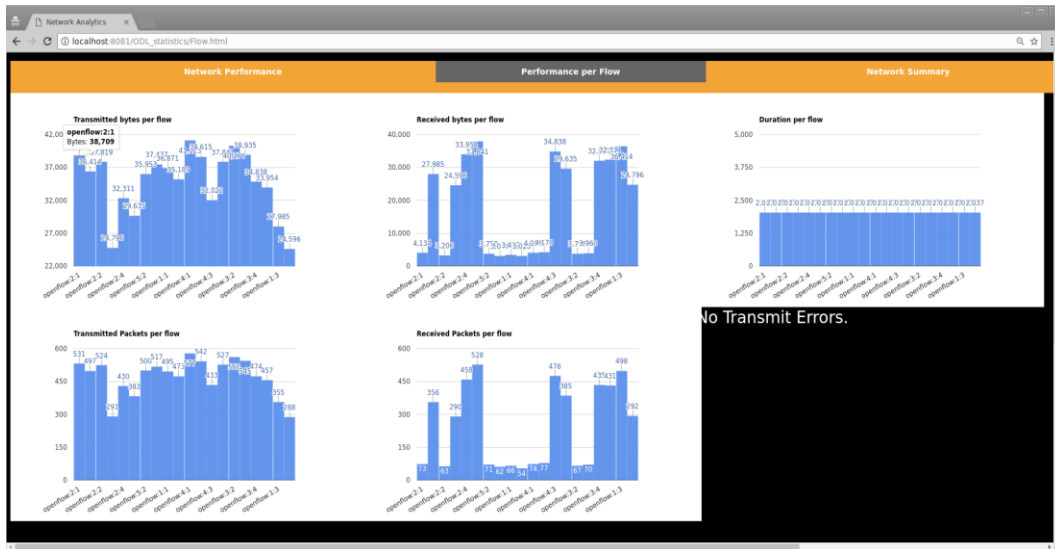


Figure 30: SDN Analytics application second page for Ring network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

The main difference between these pages is that the amount of transmitted and received bytes and packets is decreased for the performance parameters with bigger values. In addition, if we compare the duration charts, they are increased for each case but on the other hand, the amount of data is decreased.

The SDN Analytics application third pages for the linear topologies with different performance parameters are presented below:

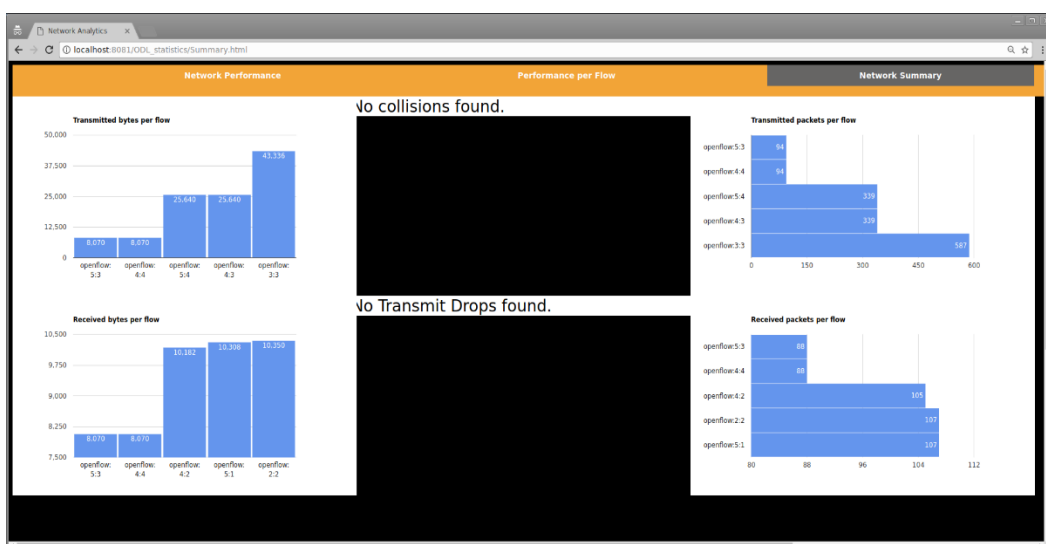


Figure 31: SDN Analytics application third page for Ring network topology without performance parameters.

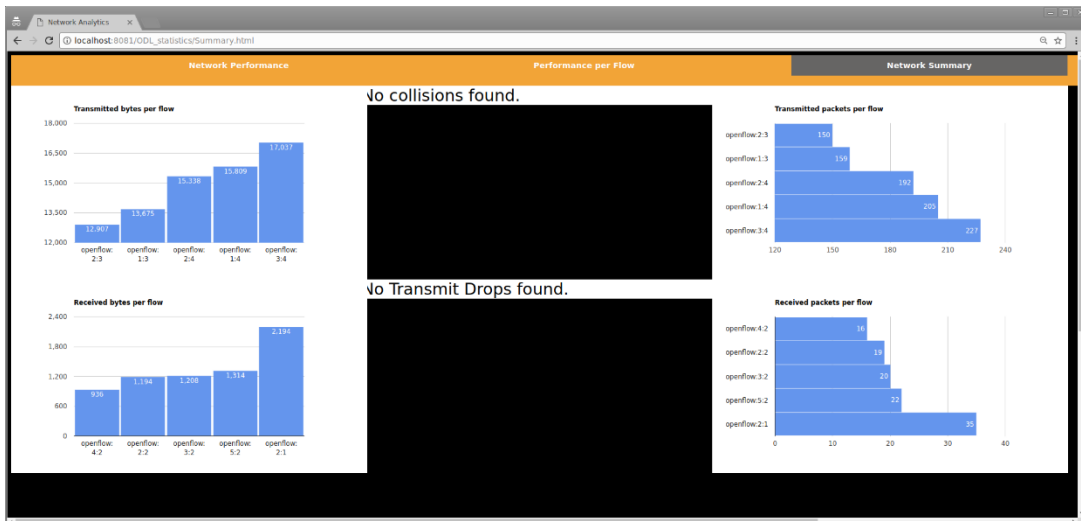


Figure 32: SDN Analytics application third page for Ring network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

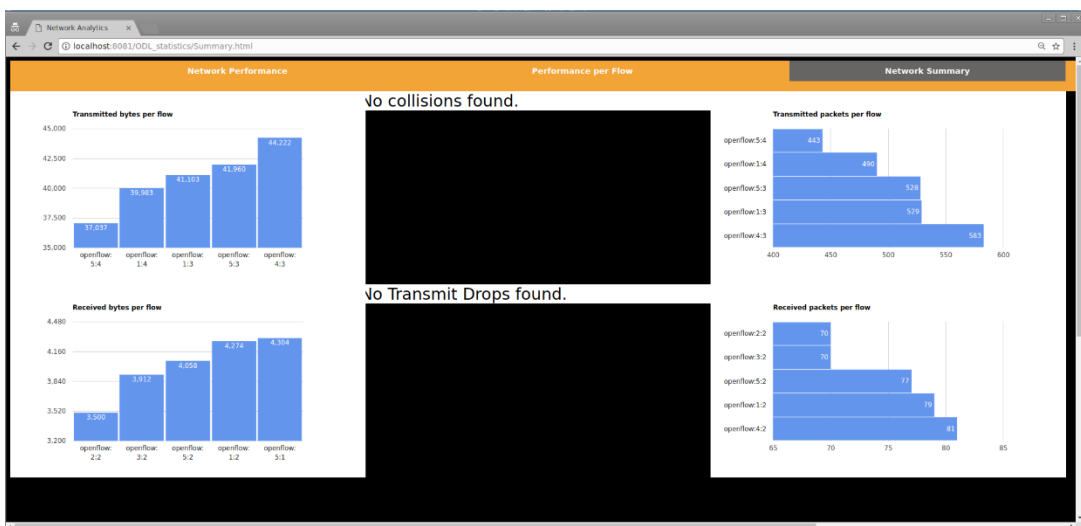


Figure 33: SDN Analytics application third page for Ring network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

In the summary page of the application. The amount of transmitted bytes and packets of the five most productive flow connectors of the topology without performance parameters is almost the same as the amount of transmitted bites and packets of the five most productive flow connectors of the topology with the second set of performance parameters. On the other

hand, there is a difference more than 50% for the received bytes and packets for the same topologies. Finally, there are no collisions or drops for any Ring topology used in our study.

### 6.3.4.3 Tree Topology Analysis

In the following pictures, the first page of the SDN Analytics application is presented for the Tree network topology. The first picture presents the total network's performance statistics without any performance parameters, In the second picture, there were added several performance parameters as we described before. They are 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss. In the third picture, the parameters are 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.



Figure 34: SDN Analytics application first page for Tree network topology without performance parameters.

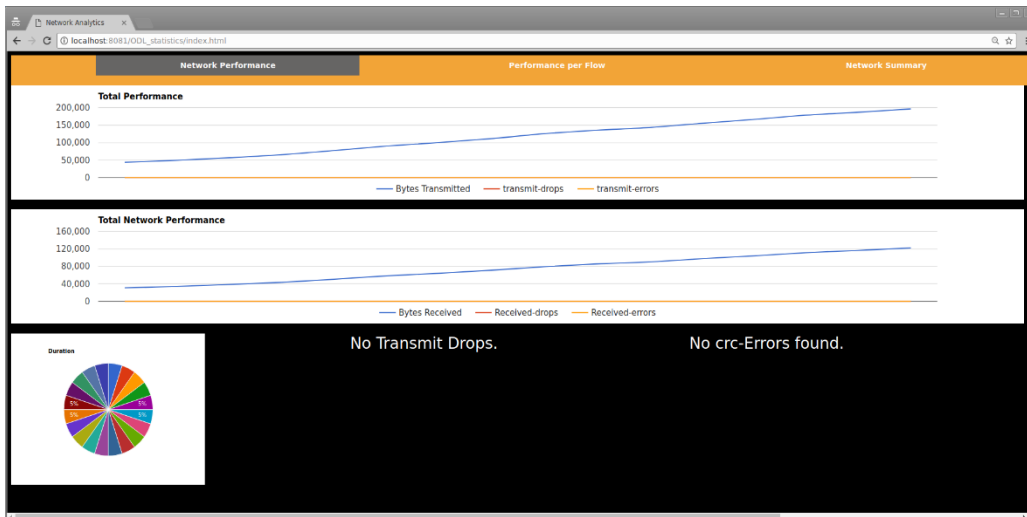


Figure 35: SDN Analytics application first page for Tree network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

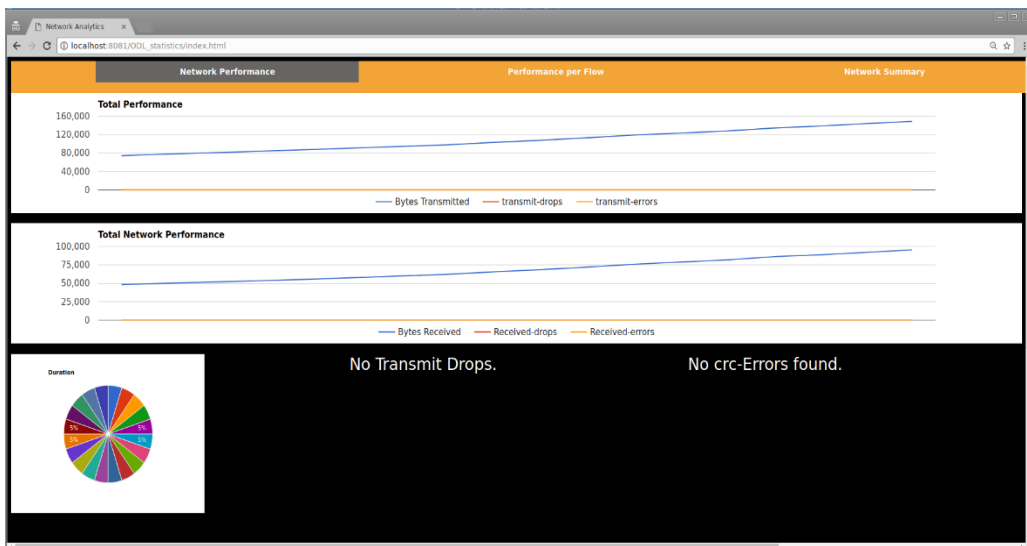


Figure 36: SDN Analytics application first page for Tree network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

The main difference between these network topologies with different performance parameters is that when traffic is generated, we are able to see the instant increase of transmitted and received bytes for the first picture. In the other two cases, the delay that was added is obvious and the amount of

transmitted and received bytes is decreased as well for the same transmission duration as it is shown in the duration pie chart.

The SDN Analytics application second pages for the Tree topologies with different performance parameters are presented below:

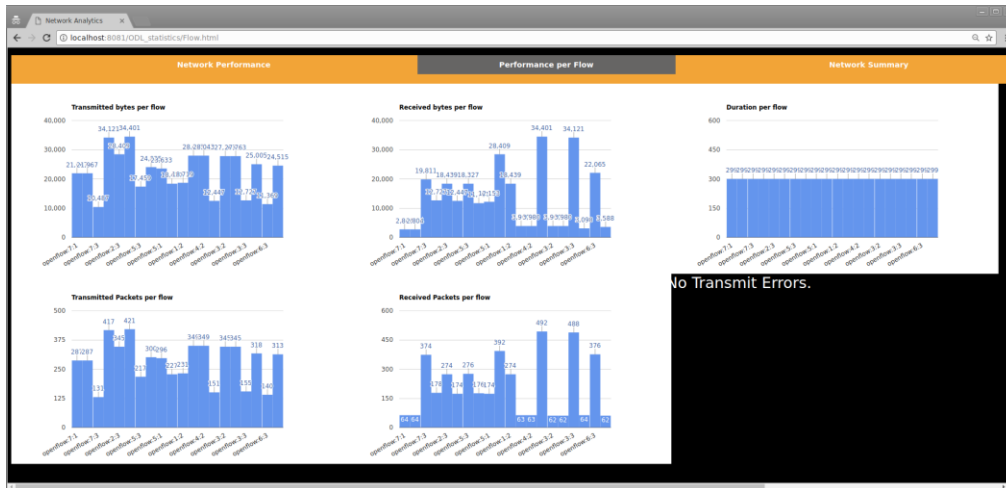


Figure 37: SDN Analytics application second page for Tree network topology without performance parameters.

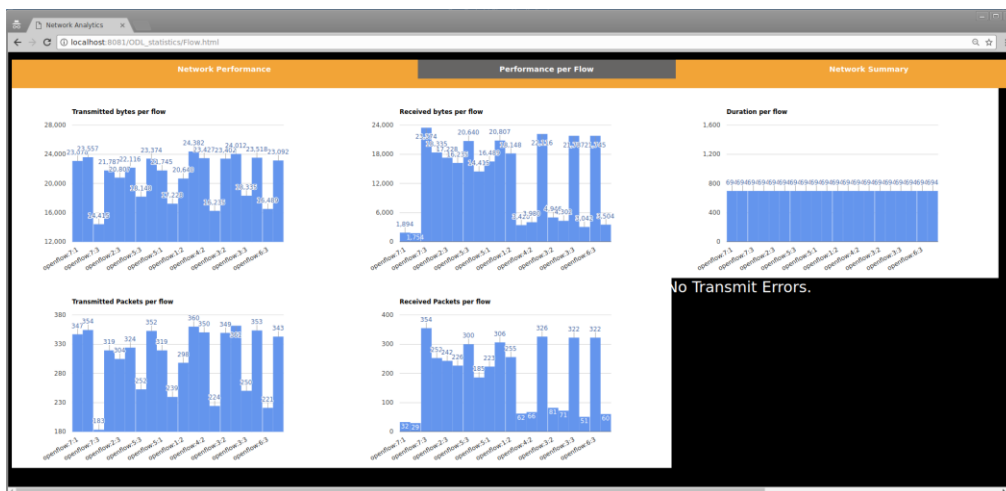


Figure 38: SDN Analytics application second page for Tree network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

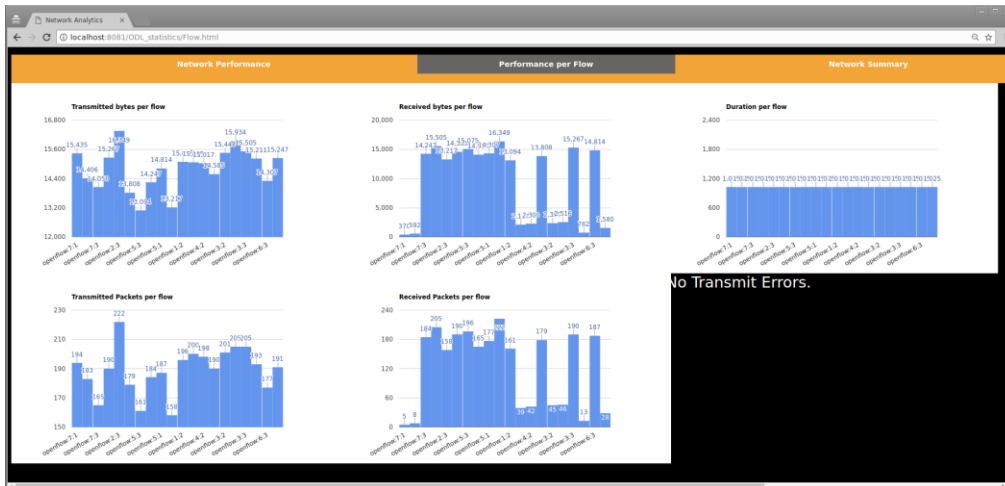


Figure 39: SDN Analytics application first page for Tree network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

The main difference between the second page of each Tree network topology is that the amount of transmitted and received bytes and packets is decreased as the performance parameters values are increased. In addition, if we compare the duration charts, they are increased for each case, although there are no transmission errors.

The SDN Analytics application third pages for the Tree topologies with different performance parameters are presented below:

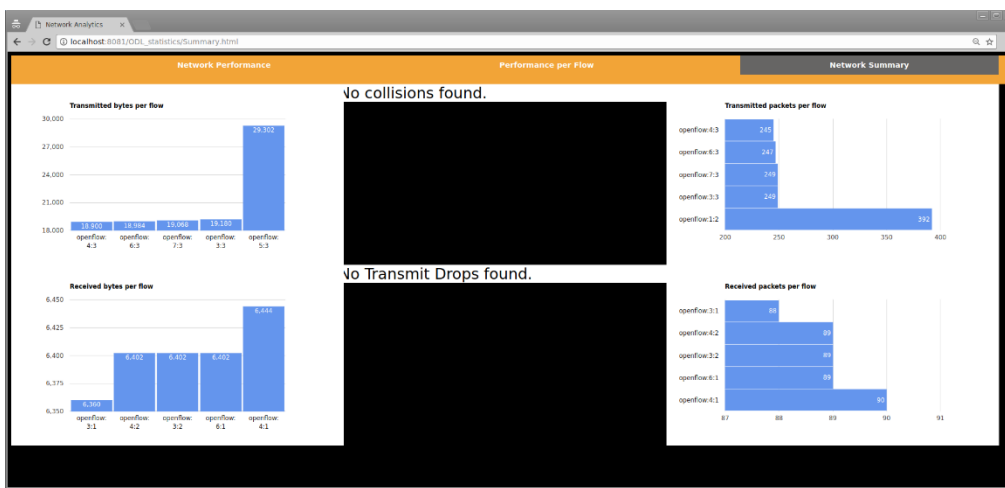


Figure 40: SDN Analytics application third page for Tree network topology without performance parameters.

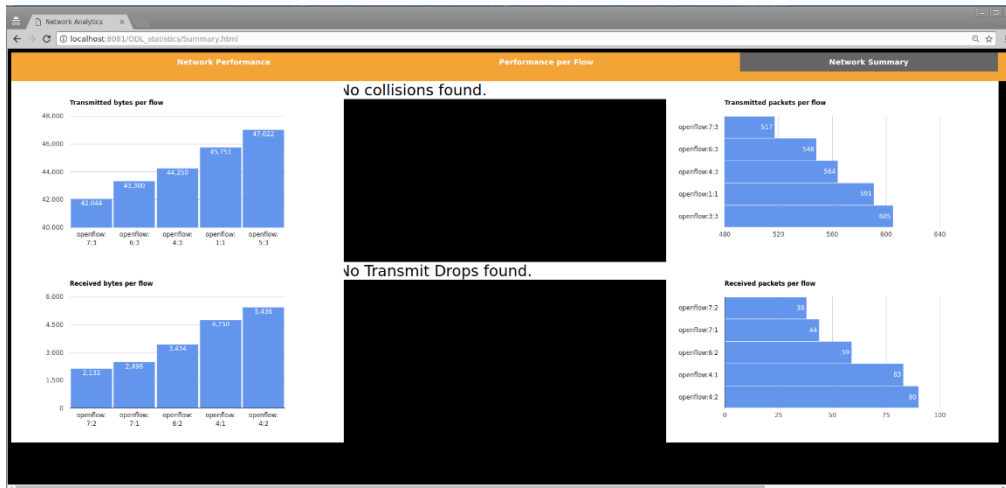


Figure 41: SDN Analytics application third page for Tree network topology with 10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss.

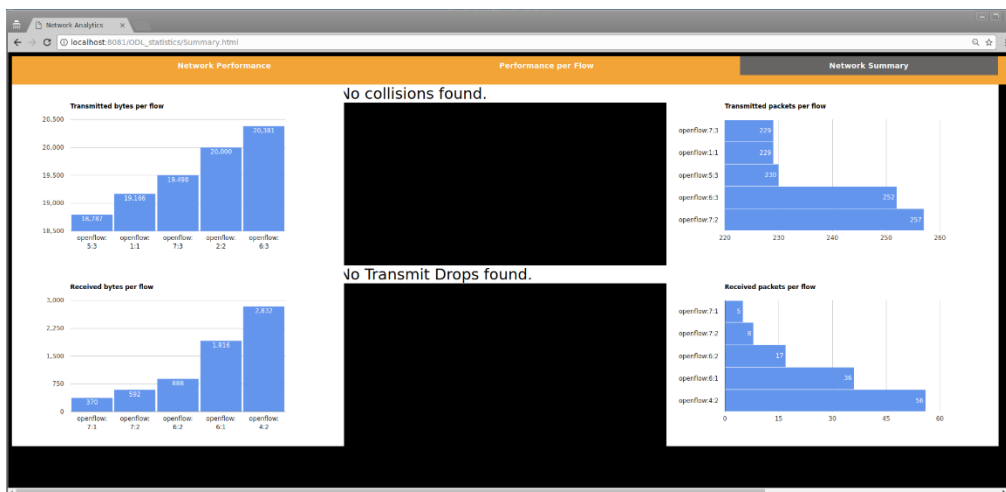


Figure 42: SDN Analytics application first page for Tree network topology with 7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss.

In the summary page of the application, there are differences to the allocation of data processing but the most active flow connectors are almost the same. On the third topology, considering the performance parameters, it is obvious that the most active node connectors are different than the other two Tree topologies. Finally, as for the other network topologies in this study there are no collisions or drops.



### 6.3.4.4 Topology Analysis Without Performance Parameters

In this section, Linear, Ring and Tree topology without performance parameters are going to be compared using the SDN Analytics application.

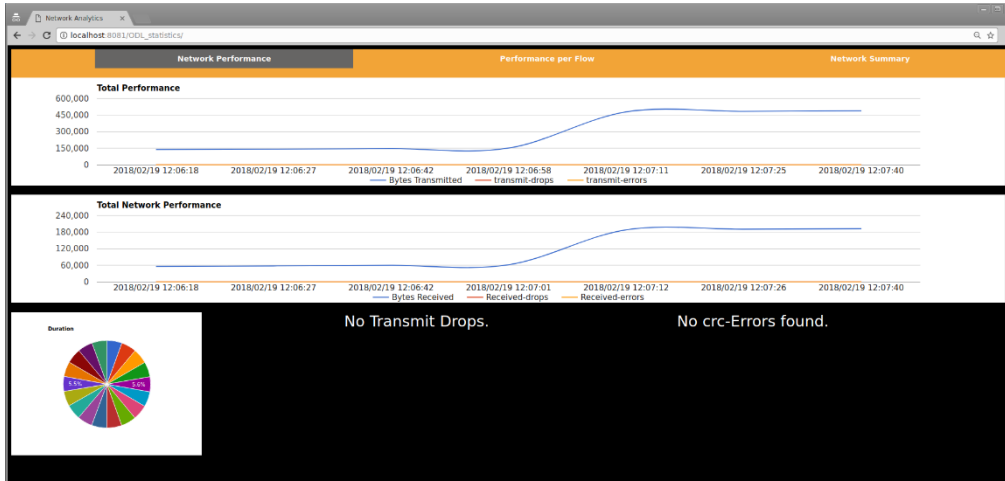


Figure 43: SDN Analytics application first page for Linear network topology without performance parameters.

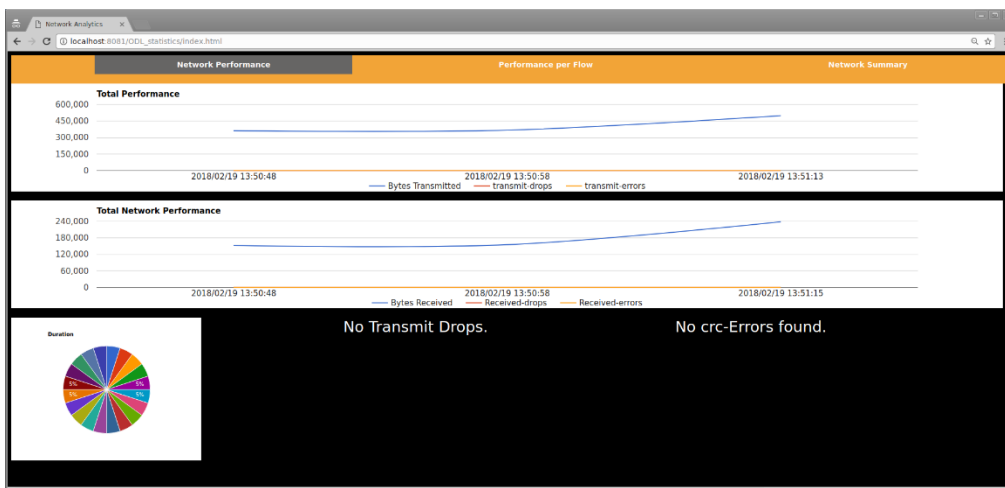


Figure 44: SDN Analytics application first page for Ring network topology without performance parameters.

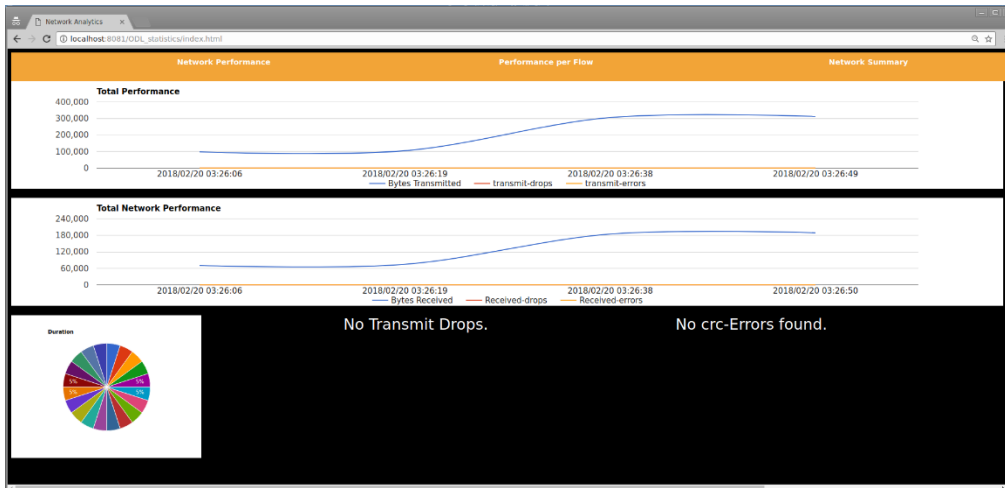


Figure 45: SDN Analytics application first page for Tree network topology without performance parameters.

As the first page of the SDN Analytics application is presented for each network topology without using any performance parameters, the total amount of transmitted and received bytes needed almost the same time to reach their destination, but for the Ring topology the transmitted and received bytes are decreased almost 50% comparing to the other two topologies. In addition, the duration of each flow connector for the linear topology is almost 1% bigger than the other two topologies.

The SDN Analytics application second pages for each network topology without any performance parameters are presented below:

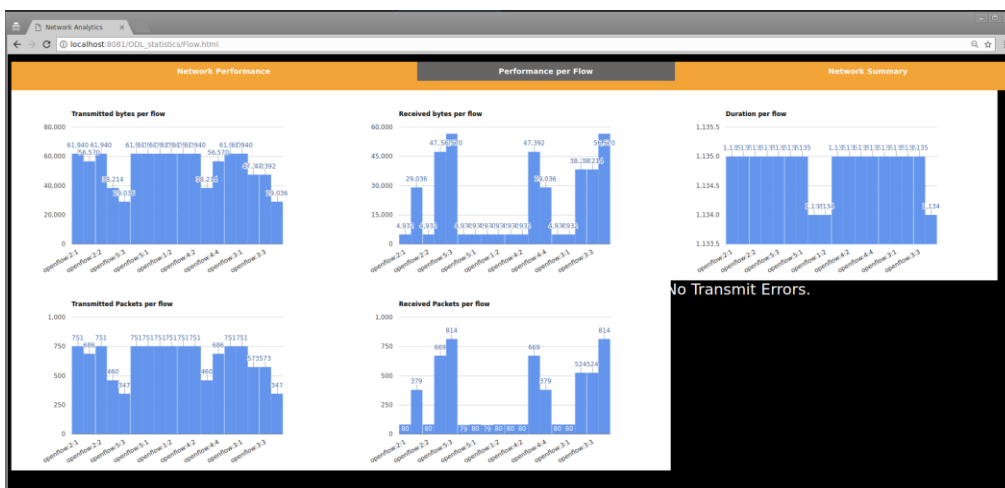


Figure 46: SDN Analytics application second page for Linear network topology without performance parameters.

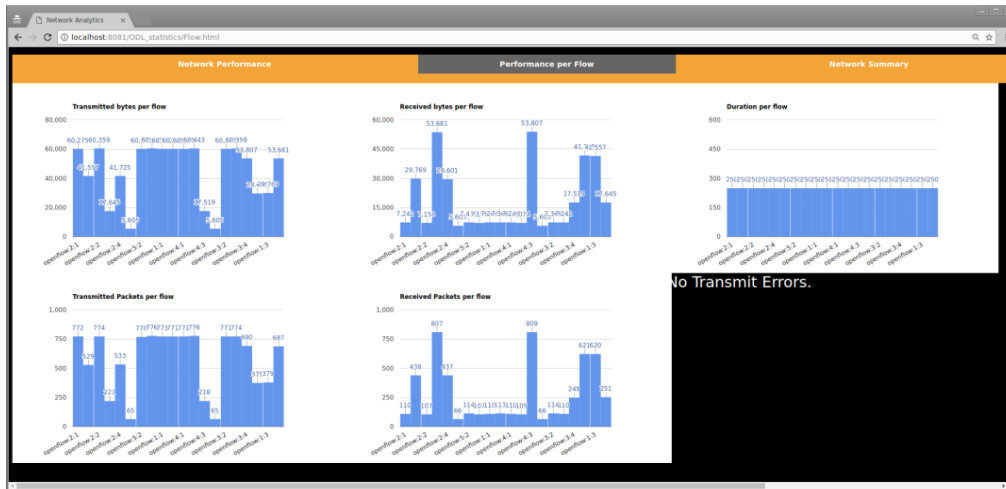


Figure 47: SDN Analytics application second page for Ring network topology without performance parameters.

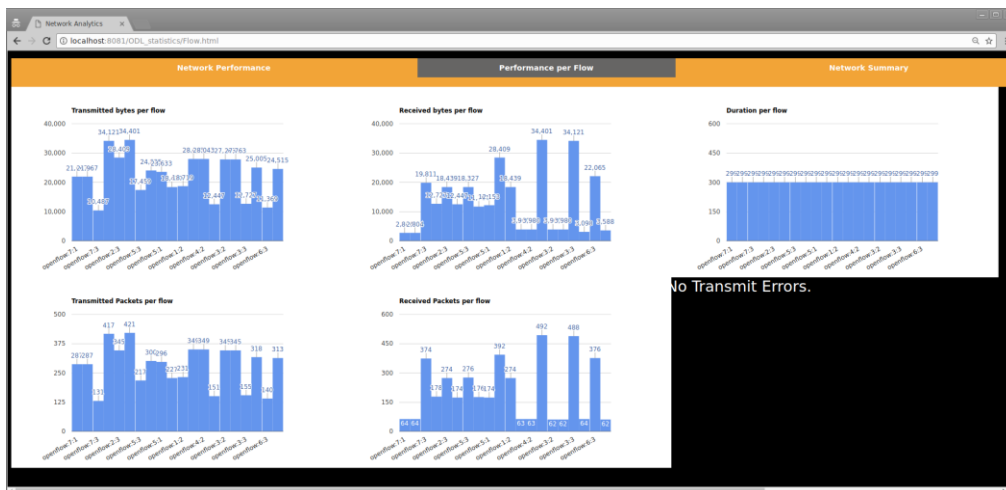


Figure 48: SDN Analytics application second page for Tree network topology without performance parameters.

For Linear and Ring network topologies, it is obvious that the second pages of the application are similar as for both topologies, all the switches are connected one after the other like a chain and the network traffic passes through each network switch. As result, the flow connectors between the

switches are more active. On the other hand, Tree network topology is more balanced, the total amount of transmitted and received bytes per flow is lower, this fact could offer improved network performance.

The third pages of the SDN Network application for each network topology are presented below:

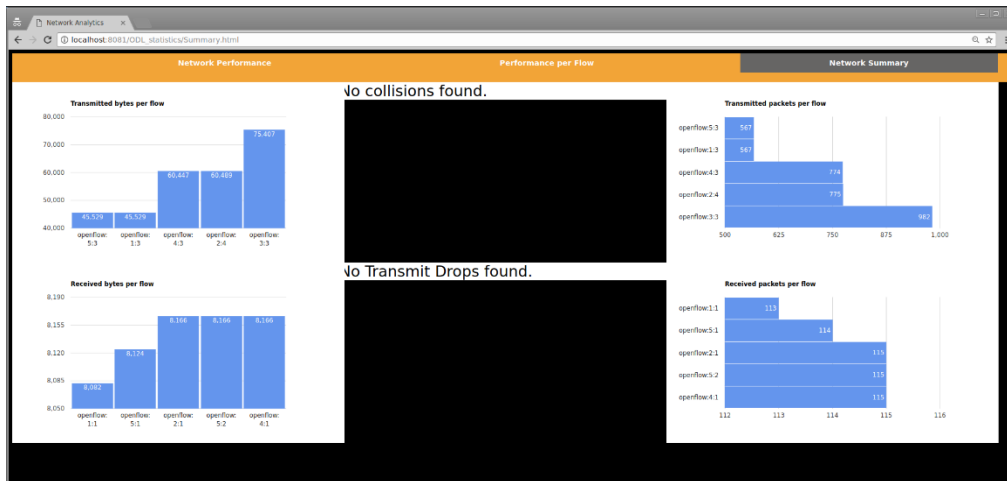


Figure 49: SDN Analytics application third page for Linear network topology without performance parameters.

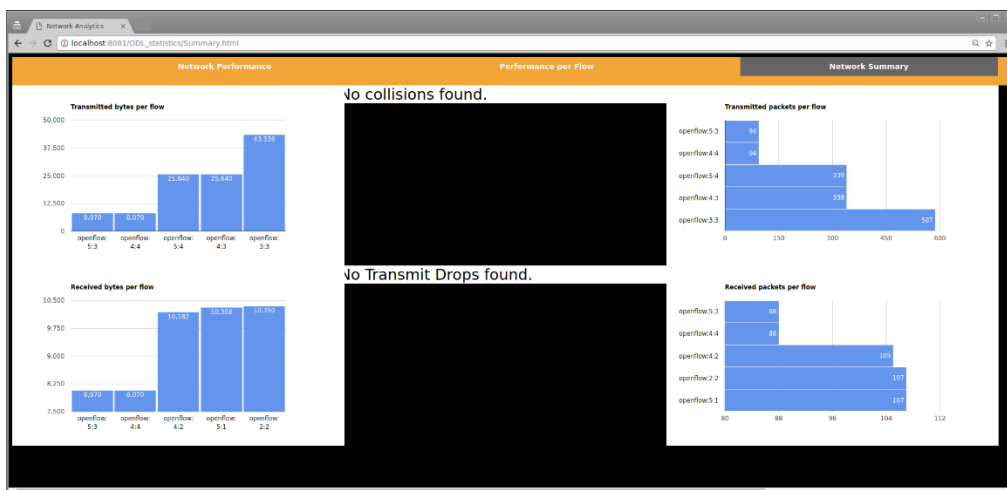


Figure 50: SDN Analytics application third page for Ring network topology without performance parameters.

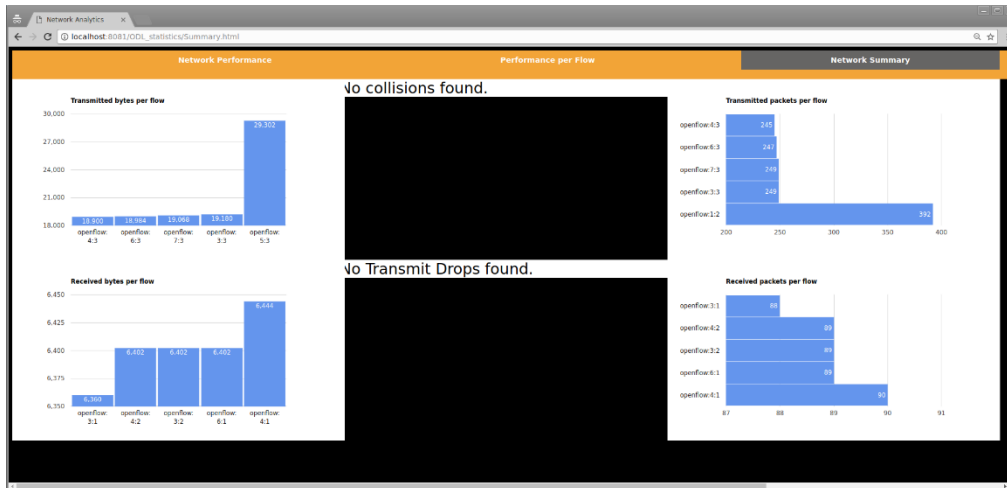


Figure 51: SDN Analytics application third page for Tree network topology without performance parameters.

The conclusion of the comparison of the SDN Analytics application third page is that Linear and Ring topologies have similar traffic flow and performance characteristics, every switch is necessary for the network but if the amount of bytes is increased it is easier to reach their maximum capacity, in contrast with Tree topology which is able to offer improved performance in larger networks as well.

#### 6.3.4.5. Topology Analysis With Performance Parameters set 1 (10 Mbit bandwidth, a maximum queue size of 1000 packets, 5 milliseconds delay and 10% loss)

The first pages of the SDN Analytics application are presented below:



Figure 52: SDN Analytics application first page for Linear network topology with performance parameters set 1.

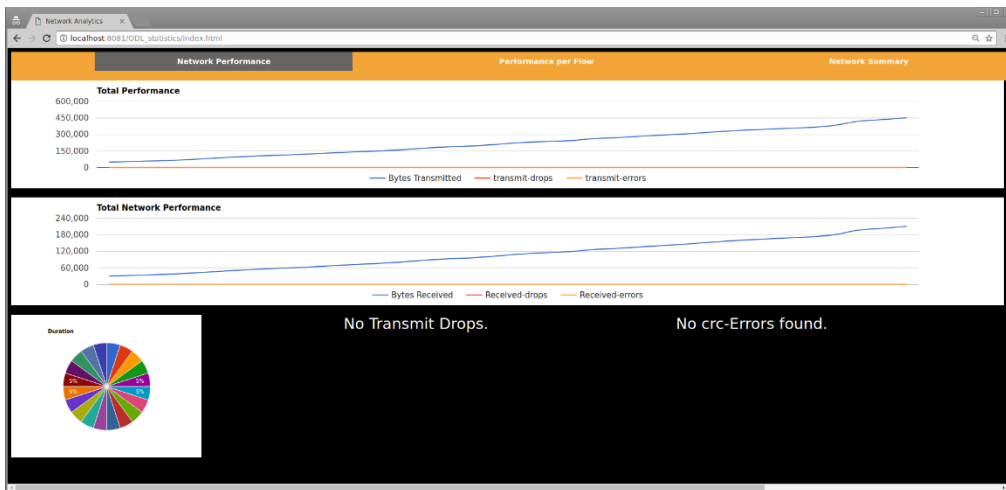


Figure 53: SDN Analytics application first page for Ring network topology with performance parameters set 1.

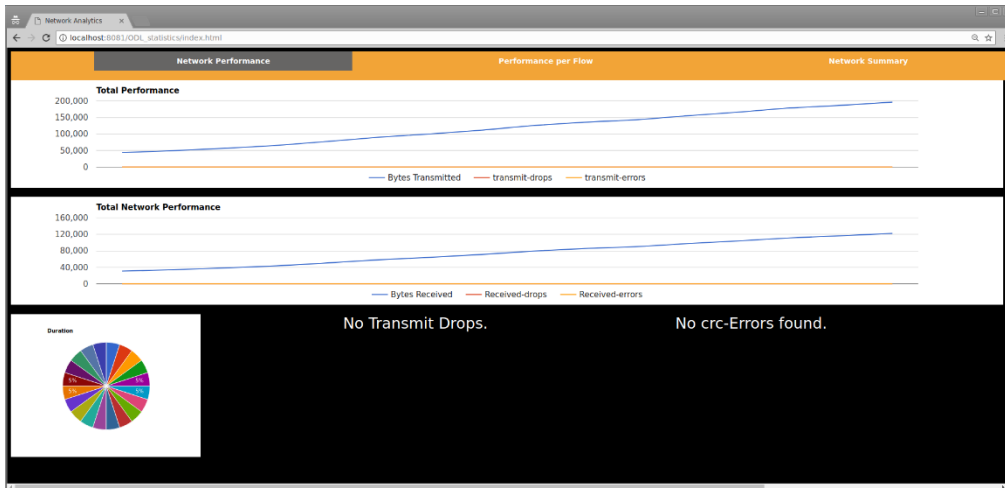


Figure 54: SDN Analytics application first page for Tree network topology with performance parameters set 1.

It is obvious here that the increased delay and loss and also the reduced bandwidth affects the network topologies. The increase of the bytes amount is slower and not constant which is due to the increased loss of some connectors. The transmission duration is almost the same for each topology and there are no drops or crc-errors.

The second page for each network topology with performance parameters set 1 is presented below:

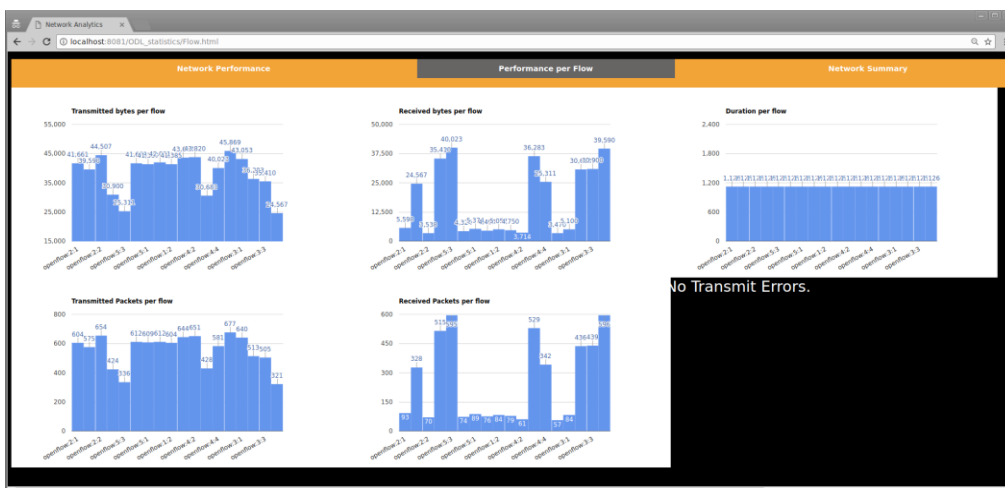


Figure 55: SDN Analytics application second page for Linear network topology with performance parameters set 1.

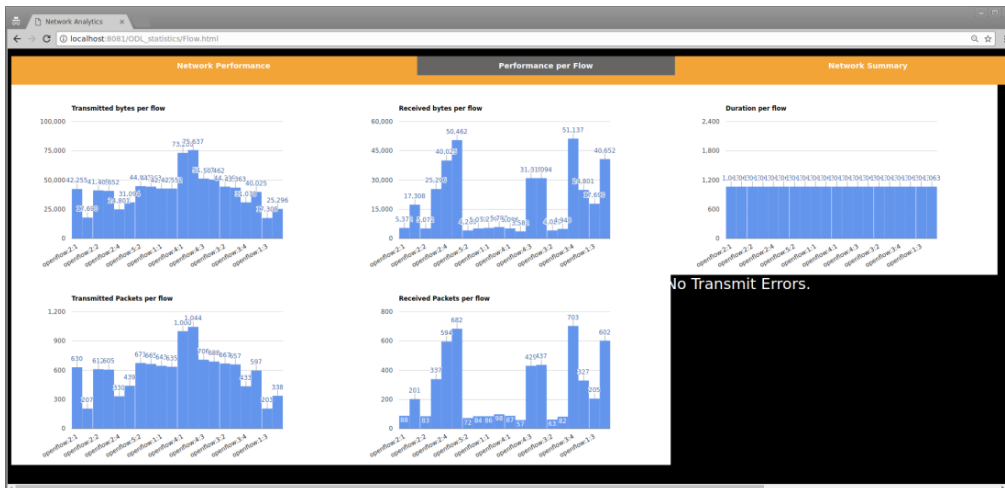


Figure 56: SDN Analytics application second page for Ring network topology with performance parameters set 1.

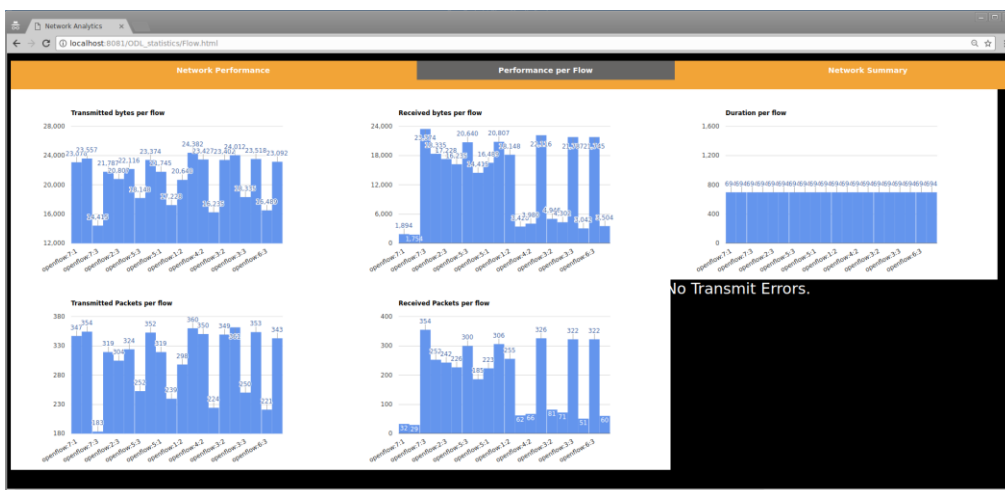


Figure 57: SDN Analytics application second page for Tree network topology with performance parameters set 1

For the Linear and Ring network topologies there are bigger differences with the performance parameters and the traffic flow is concentrated to some specific connectors which could affect the network performance. On the other hand, on Tree topology, the network traffic is divided between the connectors and despite the increased delay of the performance parameters, the duration is decreased.



The third pages of each network topologies using the first set of parameters are shown below:

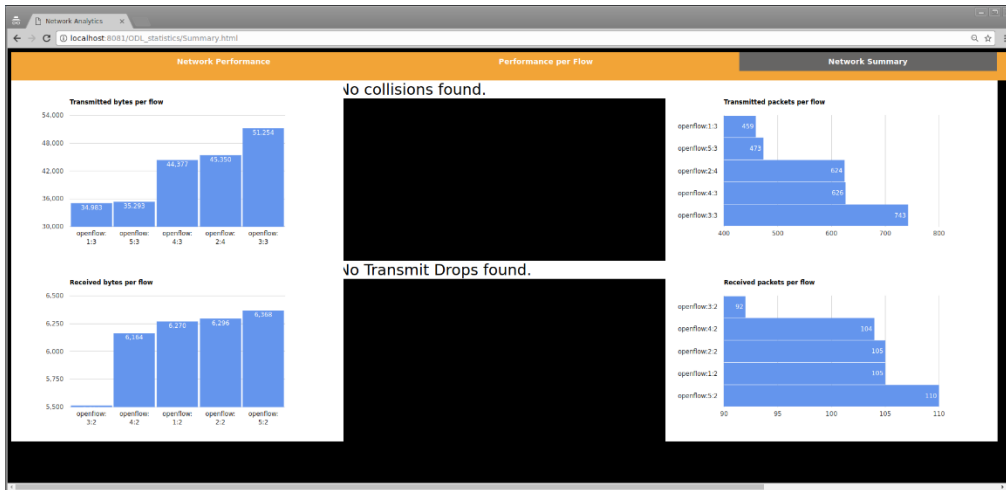


Figure 58: SDN Analytics application third page for Linear network topology with performance parameters set 1

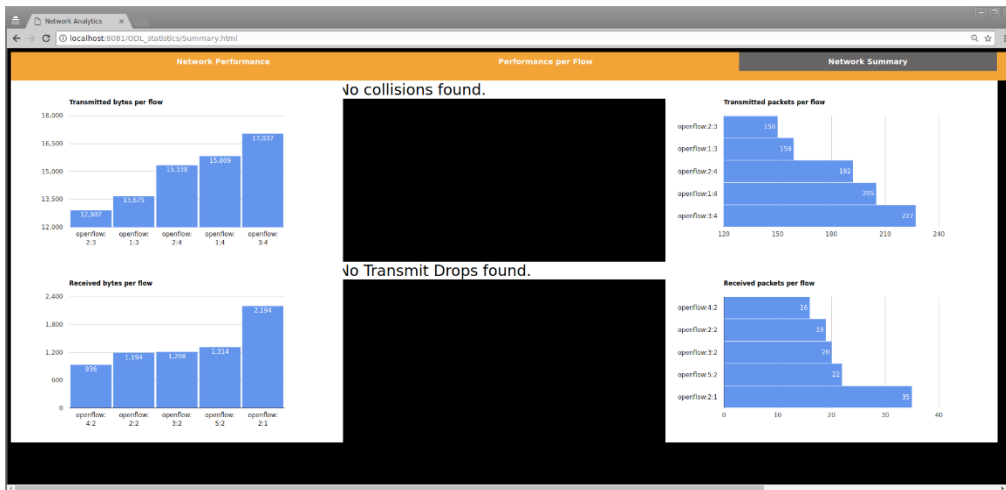


Figure 59: SDN Analytics application third page for Ring network topology with performance parameters set 1

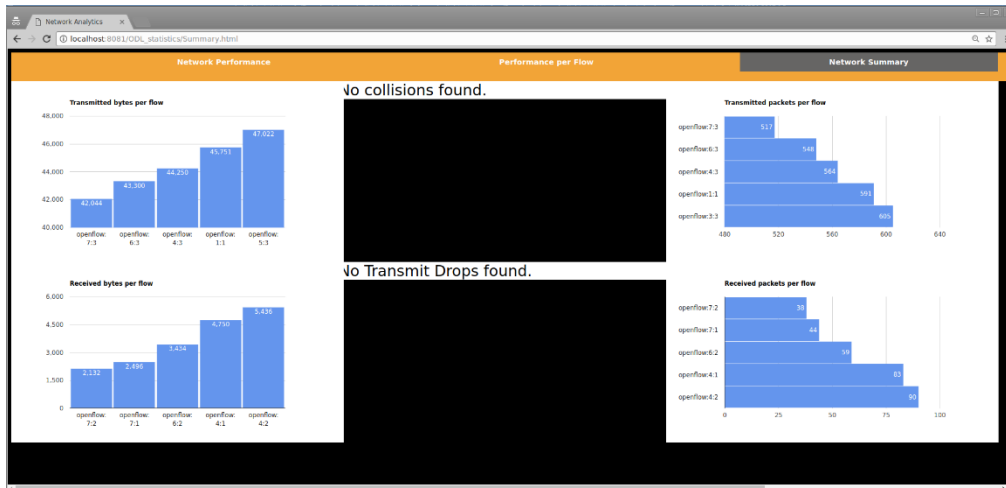


Figure 60: SDN Analytics application third page for Tree network topology with performance parameters set 1

In spite of the increased performance parameters, there are no collisions or drops for any of the topologies. In addition, it is obvious for linear and ring topologies that they are more centralized as we can see from the difference between the connector. On the other hand, in Tree topology traffic is more uniformly distributed despite the added delay and loss.

#### 6.3.4.6. Topology Analysis With Performance Parameters set 2 (7 Mbit bandwidth, a maximum queue size of 5000 packets, 15 milliseconds delay and 30% loss)

The first pages of the SDN Analytics application are presented below:

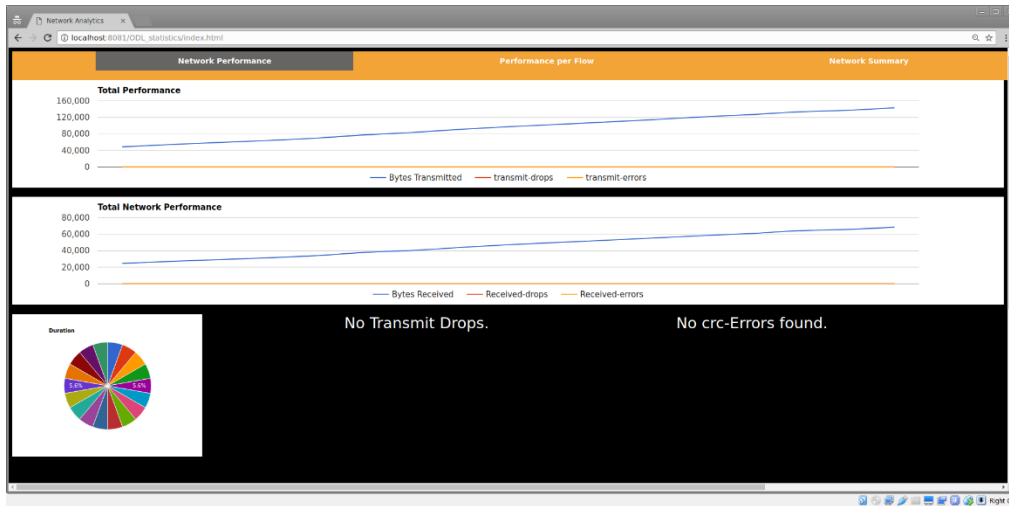


Figure 61: SDN Analytics application first page for Linear network topology with performance parameters set 2

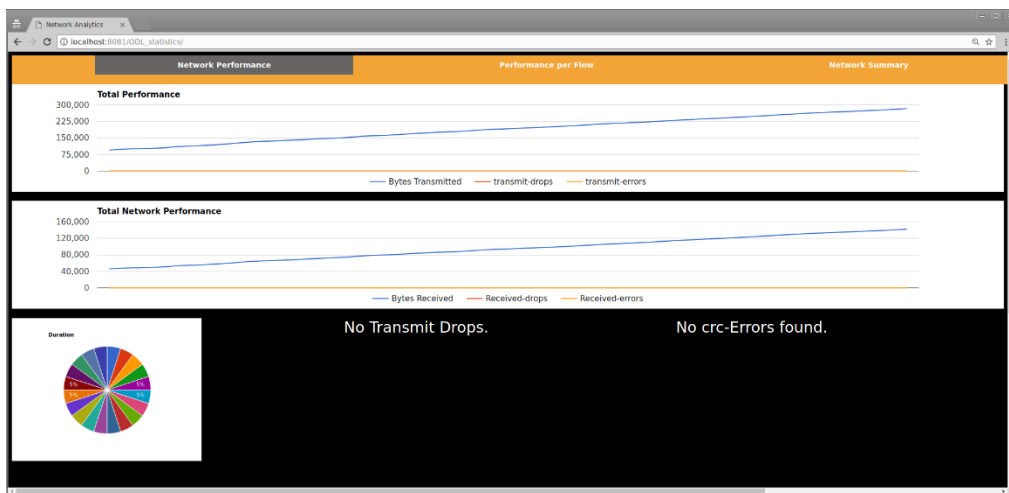


Figure 62: SDN Analytics application first page for Ring network topology with performance parameters set 2

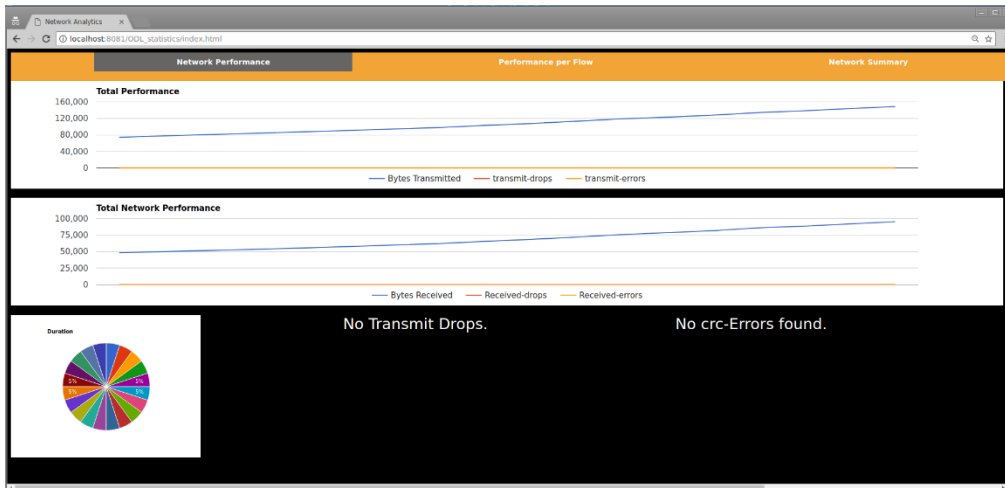


Figure 63: SDN Analytics application first page for Tree network topology with performance parameters set 2

As expected, for the second set of performance parameters with the increased delay and loss, the total amount of transmitted and received bytes are reduced for each network topology but there are no drops or crc-errors.

The second page for each network topology with performance parameters set 2 is presented below:

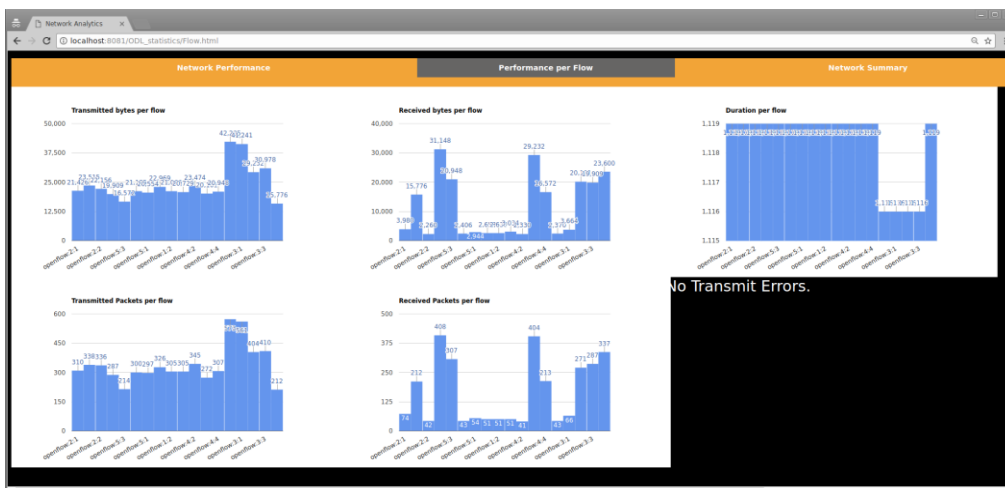


Figure 64: SDN Analytics application second page for Linear network topology with performance parameters set 2

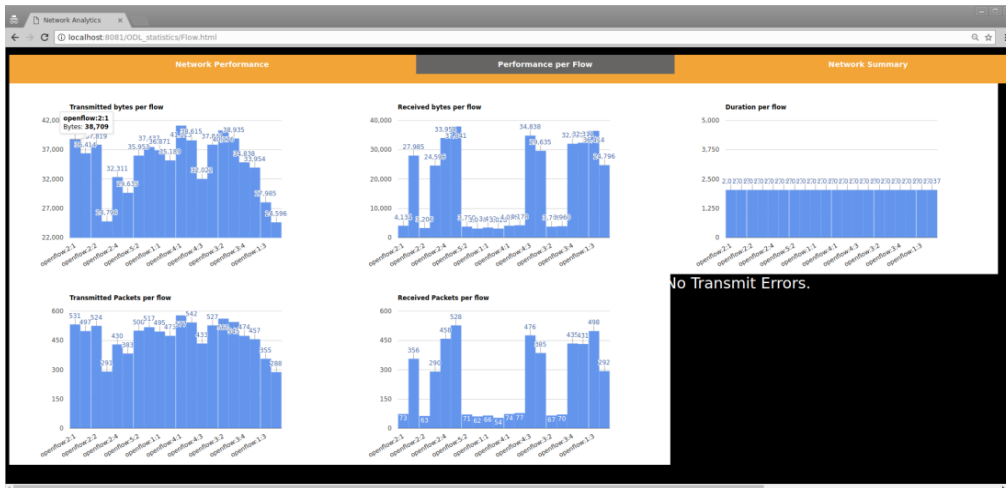


Figure 65: SDN Analytics application second page for Ring network topology with performance parameters set 2

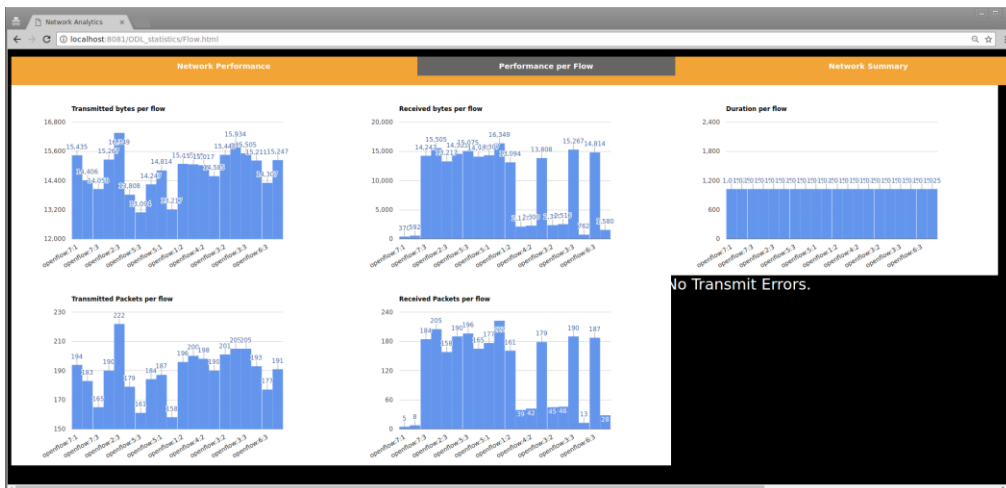


Figure 66: SDN Analytics application second page for Tree network topology with performance parameters set 2

Linear and Ring network topologies are similar for the received bytes and packets but there are also differences for the transmitted due to the increased performance parameters. On the other hand, on Tree topology, the network traffic is divided between the connectors and despite the increased delay of the performance parameters, the duration is decreased.

The third pages of each network topologies using the first set of parameters are shown below:

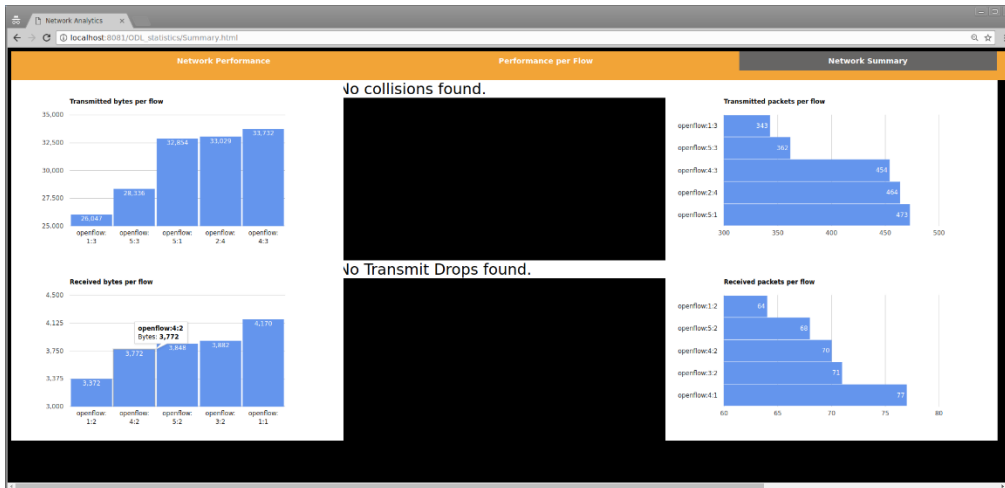


Figure 67: SDN Analytics application third page for Linear network topology with performance parameters set 2

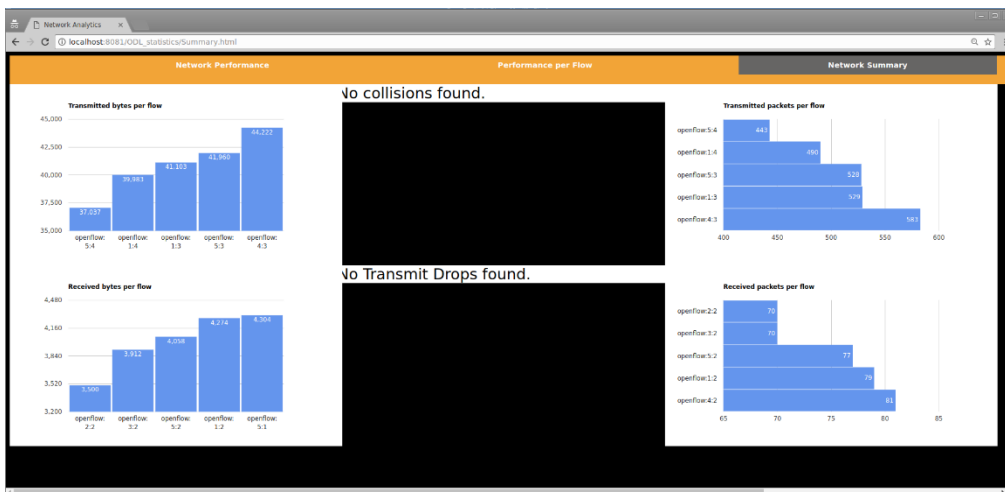


Figure 68: SDN Analytics application third page for Ring network topology with performance parameters set 2

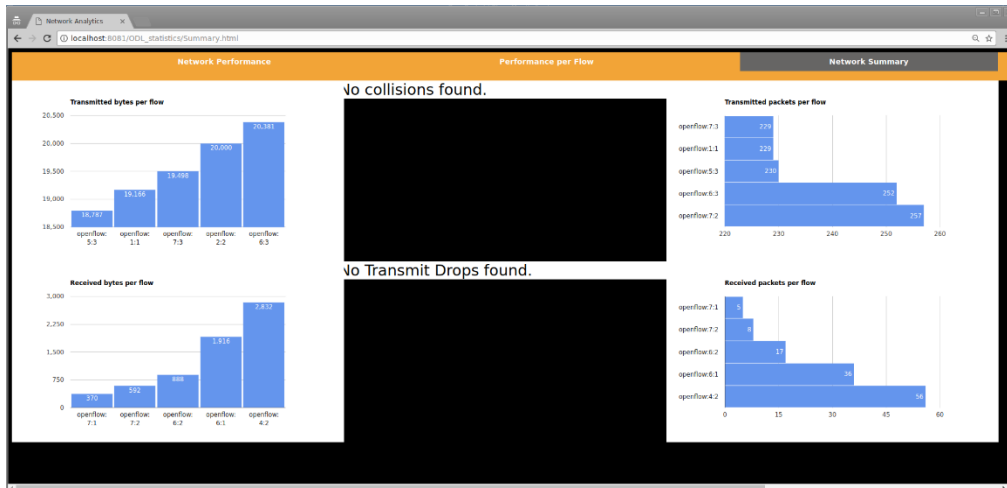


Figure 69: SDN Analytics application third page for Tree network topology with performance parameters set 2

Despite the increased performance parameters, there are no collisions or drops for any of the topologies. In addition, it is obvious for linear and ring topologies that they are more centralized as we can see from the difference between the connectors. Tree network topology is also affected by the performance parameters as we can see different characteristics from the previous tree topology cases.

## 1. Conclusion

Software Defined Network (SDN) architecture provides engineers with plenty management and control capabilities by separating control and infrastructure layers. The provided centralized network control and Openflow protocol which is compatible with the majority of SDN elements, make networks programmable and instantly configured and managed.

Software Defined Network analytics is a very useful tool used by engineers to add visibility to networks and provide them useful information about network behavior in order to prevent any possible danger for the network's stability. In other words, a Software Defined networks Analytics application provides the necessary intelligence to the network in order to help network engineers to optimize network performance and prevent any possible failures in real time.

In this study, three different kinds of network topologies were examined, Linear Topology, Ring topology and Tree topology. Linear topology is a basic topology that offers easy control and maintenance and engineers can easily detect and correct any possible errors, Ring topology is similar to linear topology but it's structure is more secure from any possible collision as the transmission is always in one direction. Finally, tree topology is a simple and balanced topology, allows simple management and configuration but it can easily become more complex by adding multiple star topologies. With the use of the Analytics application we have the necessary visibility to manage and control topologies regardless their complexity and we are able to prevent possible error collisions and other dangers in real time.

Software Defined Network Analytics application could be used as base for more SDN applications development such as a traffic forwarding application that would automatically predict possible errors and instantly avoid them.



## 2. References

1. Software-defined networking – Wikipedia  
[https://en.wikipedia.org/wiki/Software-defined\\_networking](https://en.wikipedia.org/wiki/Software-defined_networking)
2. Software -Defined Networks and OpenFlow – The Internet Protocol Journal, Volume 16, No. 1  
<https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html>
3. What's Software-Defined Networking (SDN)?  
<https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>
4. OpenDaylight  
<https://www.opendaylight.org/>
5. OpenDaylight Project – Wikipedia  
[https://en.wikipedia.org/wiki/OpenDaylight\\_Project](https://en.wikipedia.org/wiki/OpenDaylight_Project)
6. What is an OpenDaylight Controller? – SdxCentral  
<https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/>
7. Using the OpenDaylight SDN Controller with the Mininet Network Emulator  
<http://www.brianlinkletter.com/using-the-opendaylight-sdn-controller-with-the-mininet-network-emulator/>
8. OpenFlow-protocol-OpenFlow.org  
<http://archive.openflow.org/wp/learnmore/>
9. Introduction to Mininet – mininet/mininet Wiki – GitHub  
[https://www.google.gr/search?q=mininet&oq=mininet&gs\\_l=psy-ab.3..35i39k1l2j0i67k1j0.8998.10287.0.10499.7.7.0.0.0.0.124.758.2j5.7.0....0...1.1.64.psy-ab..0.7.754...0i131k1.0.1CPAlMflaIM](https://www.google.gr/search?q=mininet&oq=mininet&gs_l=psy-ab.3..35i39k1l2j0i67k1j0.8998.10287.0.10499.7.7.0.0.0.0.124.758.2j5.7.0....0...1.1.64.psy-ab..0.7.754...0i131k1.0.1CPAlMflaIM)

10. OpenDaylight DLUX:DLUX Karaf Feature – OpenDaylight Project

[https://wiki.opendaylight.org/view/OpenDaylight\\_DLUX:DLUX\\_Karaf\\_Feature](https://wiki.opendaylight.org/view/OpenDaylight_DLUX:DLUX_Karaf_Feature)

11. Chapter 2. Using the OpenDaylight User Interface(DLUX)

[https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/using\\_the\\_opendaylight\\_user\\_interface\\_dlux.html](https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/using_the_opendaylight_user_interface_dlux.html)