Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TíTULO DEL TFG: Design of an Integrated SDN/NFV Management and Orchestration Architecture

TITULACIÓN: Grau Enginyeria Telemàtica

AUTOR: Manuel Leiva Cabello

DIRECTOR: Cristina Cervelló-Pastor

FECHA: 27 octubre del 2016

Título: Design of an Integrated SDN/NFV Management and Orchestration Architecture

Autor: Manuel Leiva Cabello

Director: Cristina Cervelló-Pastor

Fecha: 27 de octubre del 2016

Resumen

Este proyecto se centra en la explicación y definición de la tecnología SDN y NFV y finalmente con la integración de los dos. También vamos a ver la lógica de esta tecnología aplicada a un programa diseñado para este proyecto.

El objetivo de este proyecto es entender para que se van a utlilizar estas tecnologías, por que compañias como Google o Microsoft llevan años invirtiendo en el desarrollo e implantación, para el uso tanto a nivel corporativo cómo a nivel de usuario.

Inicialmente explicaremos de la tecnología SDN, qué es, para que se va a utilizar y a dónde nos va a llevar. ¿Por que es tan importante SDN?
A continuación, se explicará el uso del NFV que va a cambiar la forma que tenemos de ver las redes actualmente.

Una vez tenemos claros los conceptos básicos de las dos tecnologías, vamos a realizar la explicación de la parte práctica de este proyecto.

Me gustaría comentar, que todo el software utilizado para la realización de este proyecto, es libre, desde el SO de la máquina que realizaba las simulaciones y escribía estas líneas.

La parte práctica consiste en simular cómo funcionarán los flujos de red cuando se utilice esta tecnología. El programa optimizará los recursos que nosotros queramos para un correcto funcionamiento del sistema global.
Por ejemplo, podemos optimizar el camino, el número de saltos que va a terner que realizar el flujo, que utilize el menor *delay* posible, etc.

Para finalizar obtendremos unas conclusiones específicas del trabajo realizado y unas conclusiones personales dónde se analizarán las dificultades encontradas mientras se realizaba y la formación y conocimientos obtenidos gracias al trabajo.

Tittle: Design of an Integrated SDN/NFV Management and Orchestration Architecture

Author: Manuel Leiva Cabello

Director: Cristina Cervelló-Pastor

Date: 27th October 2016

Overview

This project aims at explaining and defining the SDN technology with integration of the NFV technology.
We will also see the logic of this technology applied to a program designed for this project.

The objective of this project is to understand the purpose of this technology, where is it going to be used, why companies like Google or Microsoft for over 2 years have been investing time and resources to develop and to implement the technology on the corporate level, as well as on the level of regular user like you and me.

Firstly we will focus on the explanation of the SDN technology, what it is, what for is it going to be used and what is the future of this technology. Why the SDN is so important?
Next, we will explain the use of the NFV and show how it is going to change the way we see the network right now. NFV also works with the SDN.
Once we define the basics of the two technologies, we will proceed to the explanation of the practical part of this project.

I would like to comment on the software used for this project is free, since the OS used on the machine that realized the simulations, and wrote this lines, to the package Rstudio.

The practical part is to simulate how will the network flow when this technology is used. The program will optimize the resources that we want for the proper performance of the global system.
For example, we can optimize the path, the number of machines the flow has to cross, optimize the global delay of the flow, etc.

Finally, we will reach specific conclusions based on the work we have done, as well as some personal outcome, such as the analysis of the difficulties encountered during the performance as well as training and finally knowledge gained through work.

Table of Contents

INTRODUCTION

The technology is part of the society. In our daily tasks, everyone uses something that requires internet connection: at school, university, work…
15 years ago, it was unthinkable, but today it is a reality, and everyday we use more and more our smartphones and computers.

The smartphones and computers that we use are connected to internet. For example, this year over 1.5 billion smartphones will be sold all over the world. All this smartphones have to connect to Google Play or AppStore to download apps or synchronize. Google and Microsoft have big data centres all around the globe, but, how can we improve these massive infrastructures?

Illustration 1: Google DataCenter.

These huge companies need to optimize their infrastructure, study new technologies, and implement solutions. We have talked about smartphones, but there are computers, servers of companies, cloud computing, etc.

A few years ago, Google, Microsoft, TSPs, and other companies started to investigate new ways to optimize their networks, and ended up developing SDN and NFV technologies.

In this project, we are going to explain these technologies, their main concepts, how they work, and develop a program to optimize the resources of a given scenario.

CHAPTER 1. SDN

1.1 History

Software-defined networking (SDN) is a technology that is changing the limitations of current network infrastructures.
This technology did not appear in the last years, this was developed through a large history of innovations to make the networks more programmable.

The history of SDN begins 20 years ago, when the appearance of internet was a success, they needed to develop and manage the network properly.

At the beginnings of the nineties, older applications were replaced with modern ones. The massive use of these modern applications made researchers to design and develop new network protocols, but these new protocols needed the IETF approval, and it was too slow. So, with that issue in mind, they thought about reprogramming the network, before the conventional networks were not programmable. We could say that the SDN seed was born in those times.

In the 2000s, network traffic was increasing every year, there was a need to find more powerful and easier to manage networks, more predictable, and more confidence.
The hardware companies started to implement the logic of forwarding packets in hardware of the equipment, separate control plane. Another innovation was the centralized logic control in the network.
In that moment, the innovation was done: the distinction of the control plane and the data plane. It was standardized by the IETF, and the bases of SDN were born.

Years later, companies and universities started to investigate new architectures to have the logic control centralized. The results of these researches generated the Project 4D, consisting in 4 main layers:

-Data plane, to process the packets using configurable rules.
-Discovery layer, to know what element is in the network and traffic requirements.
-Dissemination Layer, to install packet processing rules.
-Decision layer, consisting in centralized logic controllers.

With these premises, lots of investigations started. The first was the project Ethate and its predecessor, project SANE.
Finally, Stanford University created Openflow.

1.2 Starting with SDN

The static architecture of traditional networks does not support the modern computing environments as data centres.
There are a couple of main reasons that are pushing companies to invest and evolve their infrastructure.

Year	Global Internet Traffic
1992	100 GB per day
1997	100 GB per hour
2002	100 GBps
2007	2,000 GBps
2015	20,235 GBps
2020	61,386 GBps

Source: Cisco VNI, 2016

Illustration 2: Historical benchmark for total internet traffic. Source: Cisco

-As we can see in the image above, global internet traffic is increasing really fast.

--In 2002, we can see that it increased from 100GB per hour to 100GB per second.

--In 2007, the traffic increases every month, and the predictions say that it will grow exponentially.

This huge increase of the traffic is caused by the streaming. Nowadays, most of the retransmissions are in the conventional TV and in internet streaming.

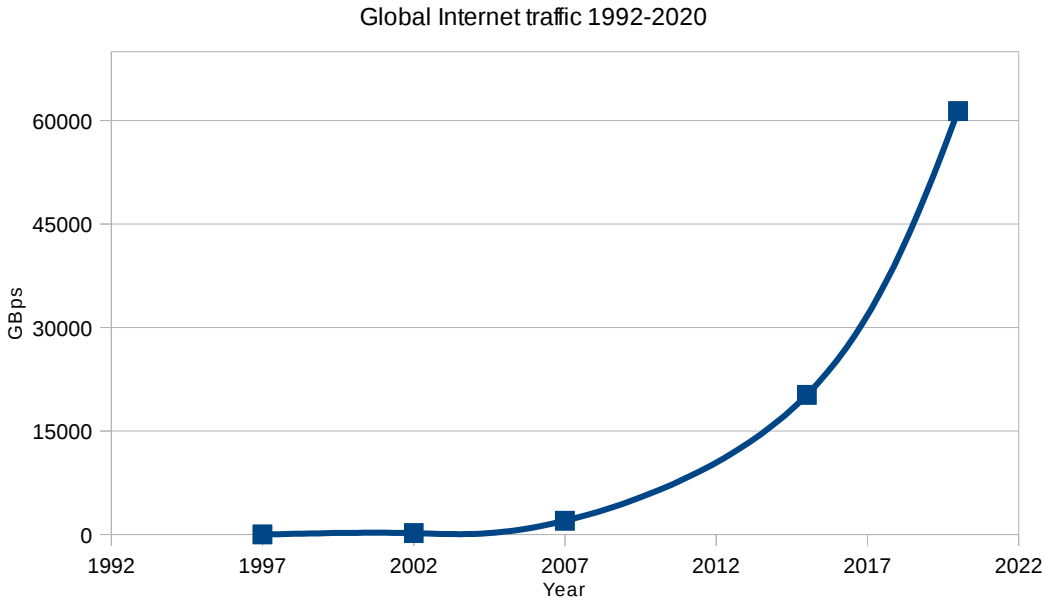

Illustration 3: Representation of Illustration 2 in graph

1.2.1 Why implement SDN?

Now, it is time to expose reasons why “big companies” are expending resources developing SDN solutions:

-Today there are huge companies as IBM, Google[1], Amazon or Microsoft[2] offering that are offering different cloud services, this situation induces many companies to not invest in physical servers. They prefer to pay for these cloud services, so the infrastructures of data centres offering that are growing every day, and it can be a problem to manage the all the traffic that the infrastructure has to support.

- Big data is a reality. Thousands of databases are being analysed, so it requires the massive processing power of thousands of servers. This is creating a demand of bandwidth.

- Traffic patterns are changing. A few years ago, apps were client-server applications; nowadays everything is much more complicated. There are different connections to databases and servers. Now, anyone can connect from anywhere, at any time, with his smartphone, tablet or laptop, generating lots of traffic crossing the WAN every second.

Illustration 4: Global devices and connections growth. Source: Cisco

**CAGR: Compound annual growth rate*

1.2.2 Main SDN concepts

Networks defined by SDN[3] had a network architecture that has dynamism, adaptability and profitability.

New architectures that provide this technology are going to push the technological companies to the next level of cloud computing and cloud-based services.

First we have to define a couple of concepts:

-The control plane: Is the logical part of the router, with the protocols that are being used in the network. Control plane is going to manage the routing tables.

-The data plane or user plane: Defines the part of the router's architecture that decides what to do with packets arriving on an inbound interface. Normally, the router looks into a table to see the destination address of the incoming packet.

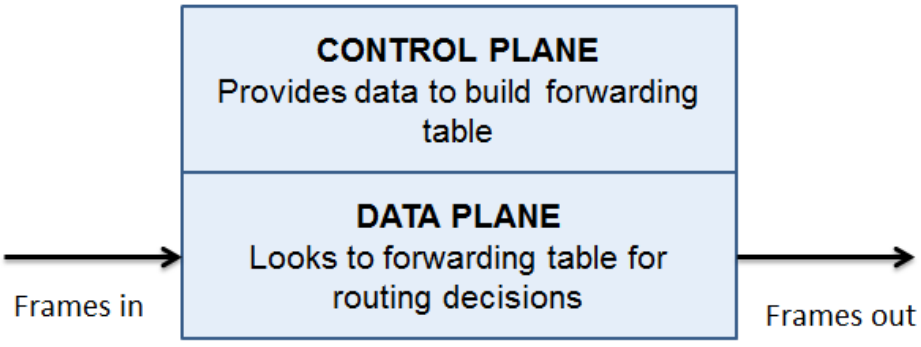

Illustration 5: Control and Data plane graphic

1.3 SDN Infrastructure

1.3.1 How can SDN improve current network infrastructures?

There are a couple of SDN properties that help in the objective of improve current network infrastructures.

- SDN breaks the vertical integration, separating the network's control logic (control plane) from the routers and switches that forward the traffic (data plane).

- Second, with the separation of control plane and data plane, network switches become simple forwarding devices, and the control logic is implemented in a centralized controller.

SDN controller, or NOS (Network Operation System), is the brain of the network, the strategic point of control, where it interconnects the data plane with control plane.
The controller generates network configuration based on the policies that has defined.
NOS has the topology network information, so it can do a device discovery and distribute changes on the network.

1.3.2 Centralized or distributed controller

There are 2 ways to implement controllers in SDN network, the centralized controller or distributed controller[4]:

-A centralized controller is an entity that manages the entire network and all forwarding devices. This can be critical due to the reason that it is a single point of failure: if this node is down, all the network will be in trouble. This can happen for a lot of reasons, like a hacking attack, such as a DDOS (Distributed Denial of Service) or a large number of data plane elements that could induce a bottleneck if the controller can't handle the load.

-A distributed controller or network operating system, the main point is that it can be done as a cluster of nodes.

This idea of distributed controllers offer weak consistency semantics, which means that data updates on distinct nodes will eventually be updated on all controller nodes. This implies that there is a period of time in which distinct nodes may read different values (new value or deprecated value) for a same property.
For example, if a big provider has different data centres all around the world and one controller of one data centre is down, we could have problems.

1.3.3 Northbound and Southbound API

These interfaces are useful to interconnect SDN controller to the network elements.

1.3.3.1 Southbound API

In southbound API[5], the controller can make dynamic changes into the network switches, if it is required.
The most common interface between controller and the network elements is Openflow, which is developed by ONF (Open Networking Foundation)
Another different example is Cisco OpFlex.

1.3.3.2 Northbound API

The northbound interface[6] can be used for innovation, organization, and automation of the network. The could be many different applications there and it is a sensitive point. It is developed by the Open Networking Foundation.

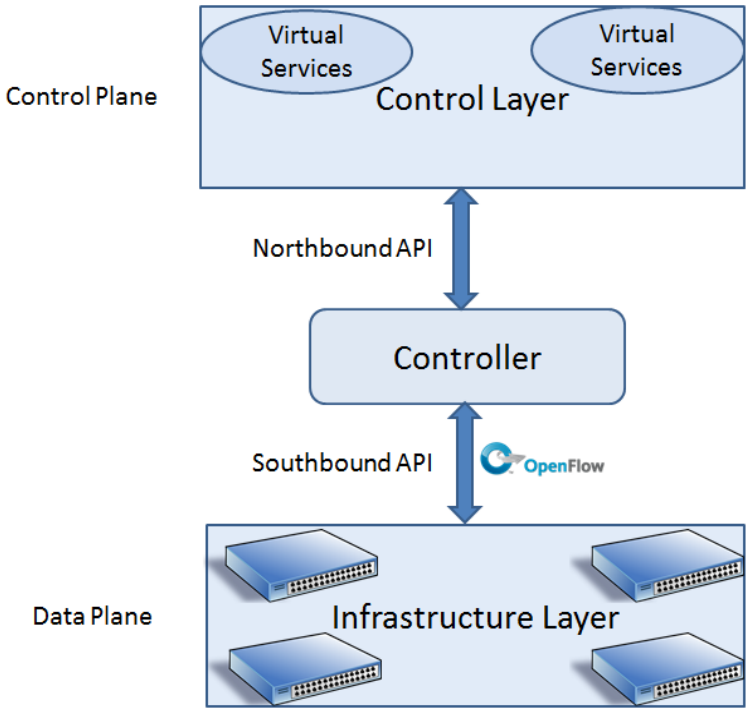

Illustration 6: SDN infrastructure

CHAPTER 2. NFV

2.1 NFV - Network Function Virtualization

2.1.1 History and explanation

Network Function Virtualization[7] was created by a consortium of service providers.
Service providers tried to speed up deployment of new network services in order to increase their revenue and growth plans. They looked to standard IT virtualization technologies and found NFV helped accelerate service innovation.

Network Function Virtualization is an architecture concept based on virtualizing some functions made by the network. So, NFV does not need specific hardware to run a particular software to define network functionality.
What NFV proposes is to use high number of servers, switches and storage, to virtualize functions with hypervisor and install the different software that is going to make the network operations.

There are few points to consider with NFV:

- Distinguish between software and hardware. In the development side, it is a superb option, because each component can run regardless of the other.

- With high volume servers that have multiple network functionalities virtualized, the infrastructure resources can be reassigned if needed.

- With NFV, the infrastructure will have a great level of flexibility, and the performance will be dynamic and finer granularity.

Example:

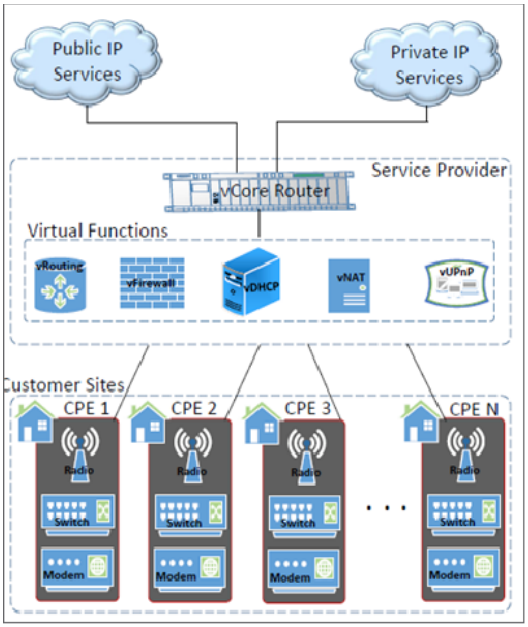

Illustration 7: Service Provider Infrastructure

In the Illustration 7, we can have look at providers' infrastructure with NFV architecture applied.
On the customers' side, we can identify only a modem, a switch and a Wi-Fi antenna.
On the service provider's side, we can see all the services that can be virtualized. For example, firewall, DHCP or routing.
So, if the service provider is going to implement in their high volume servers most of the services, customer's system will remain simple, meaning that even if the service provider implements new functionalities, customer will, most likely, not be affected.

2.2 NFV Issues to be resolved

NFV technology has some issues that have to be resolved. It is a new technology and needs some years of development.

- The performance has to be the similar if there is a NFV implementation or specific software.
This implies that all the potential bottlenecks and layers have to be evaluated and solved.

- The functions or services of the different users should be protected from each other.

-The physical and virtual resources should be protected from the subscribers. One way to secure them is using internal firewalls and, if something happen, it will impact the lowest amount of users.

- Reliability and availability are not only a customer expectation, but often a regulatory requirement, as TSPs are considered to be part of critical national infrastructure.

- The main point of NFV is based on breaking the barriers that result from proprietary hardware-based service provision.
Any NFV platform must be an open shared environment, and capable to run different applications from different companies. Infrastructure providers (InPs) must be free to make their own hardware selection decision without any hardware-software compatibility problem.

- NFV needs to be scalable, so it will be able to have millions of users at the same time without any complication.

2.3 SDN and NFV together

Software-defined-network and network function virtualization[8] are highly complementary. The two concepts and solutions combined can give great potential.
These two technologies can be applied separately, they are not dependent.

Considering this two technologies working together, we are going to see some figures that explain the situation.

2.3.1 No NFV and SDN implemented

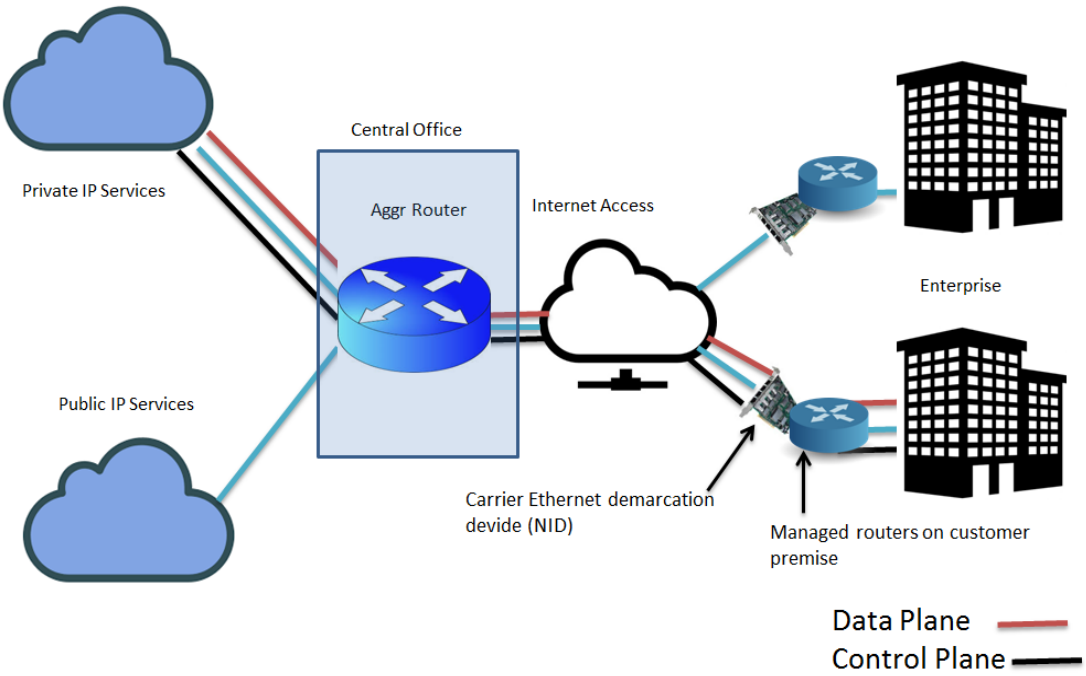

Illustration 8: Traditional infrastructure

In illustration 8, we can see the traditional infrastructure that everybody knows, without any of the technologies we have explained: a company that connects to the internet crossing its router and firewall and then the central office with an aggregation router.

2.3.2 NFV implemented

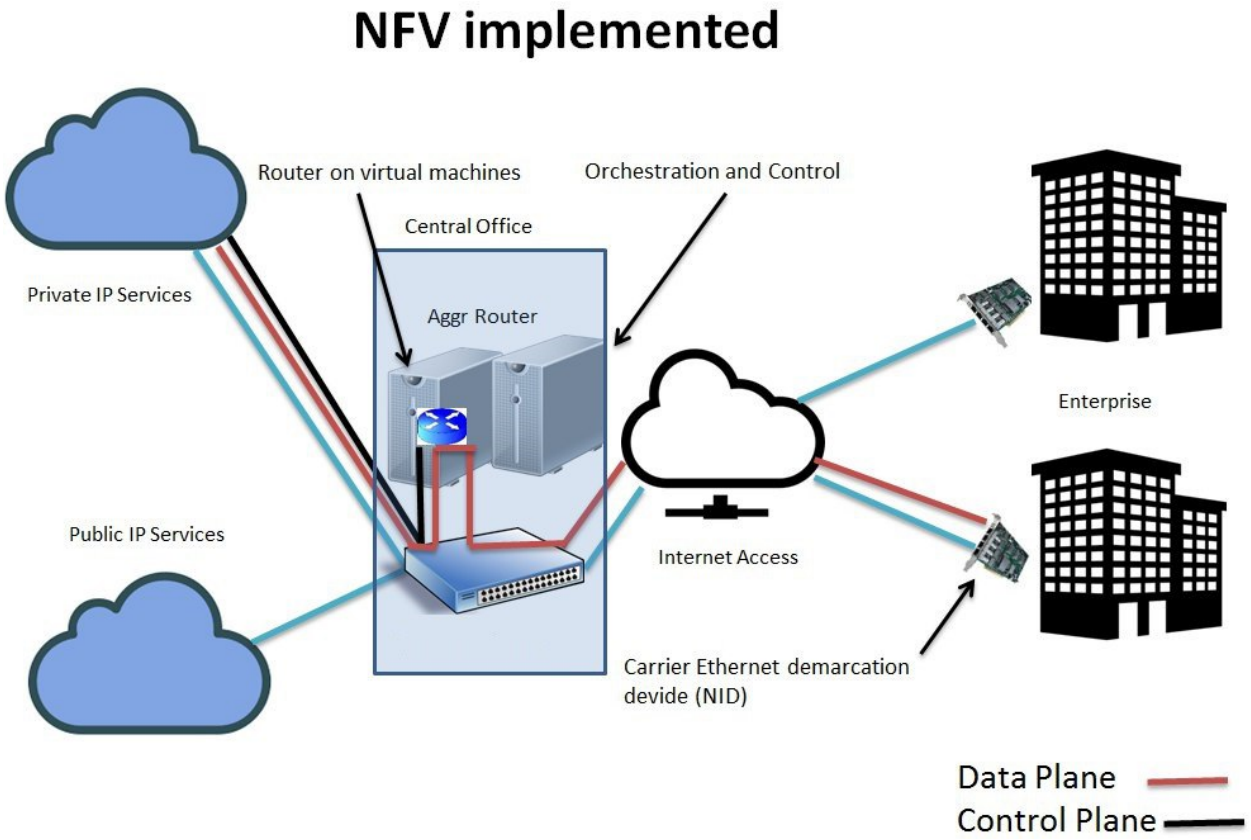

Illustration 9: Infrastructure with NFV implemented

In the illustration 9, we have NFV implemented, so there are a couple of changes.

- Now we have virtualized the router function and all that is left at the customer side is a Network Interface Device (NID) for providing some information of the client, performance, etc.

2.3.3 NFV and SDN implemented

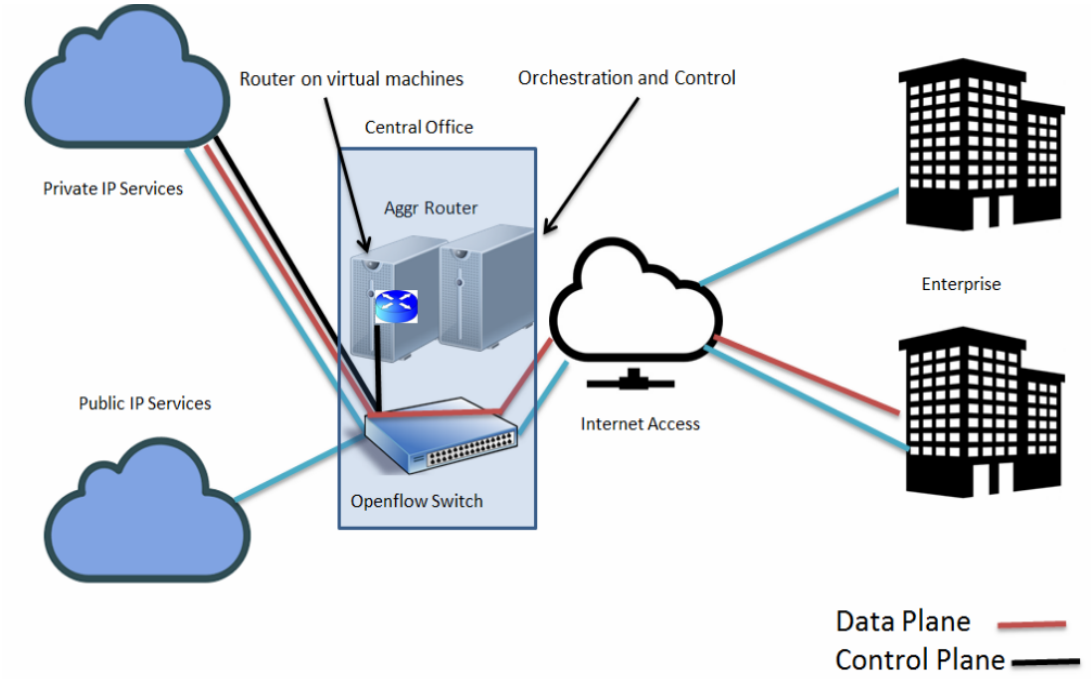

Illustration 10: Infrastructure with NFV and SDN implemented

Finally, in the illustration 10, we have the SDN and NFV solution:

The control plane and data plane are separate. Now, the data packets are forwarded by an optimized data plane, and the control plane is running in a virtual machine with generic hardware.

2.3.4 Conclusion

To conclude, the combination of SDN and NFV gives us an optimal solution:

-The expensive and dedicated hardware is replaced by a generic hardware with specific and dedicated software.

-The control of the data plane has been separated and, if one of the two parts needs an upgrade, it is not going to affect the other part. The upgrades of the software are easier, and are the same for the generic hardware.

-Better optimization of Capex and Opex.

CHAPTER 3. LINEAR PROGRAMMING

3.1 Presentation

The practical part of this thesis consists in the optimization of some parameters in a network. This imaginary scenario will be implemented with SDN and NFV.
First, we will explain the variables that we used in the program, and then we will explain the implemented formulas.

We used GLPK-GNU Lineal Programming kit, which is explained in the annex [9].

The variables used:

$G(V, E)$ → Topology, V are the nodes, E are the edges.
$s \epsilon V$ → Source belongs to nodes.
$d \epsilon V$ → Destination belongs to nodes.
$l \epsilon E$ → Links belongs to edges.
$\lambda[i, j]$ → Delay from i to j.
$L_{s,d}$ → Set of sources/destination requesting traffic.
$C_{s,d}^{p}$ → Path chosen between s and d.
$D_{s,d}^{k}$ → Demand k from source to destination.
i_{s}^{k} → if I = source [k] → 1
i_{d}^{k} → if I = destination [k] → 1
$\lambda_{s,d}^{v}$ → Process delay of the node
$vfn[i, j]$ → If the path selected cross NFV is 1 if not is 0

3.2 Functions

3.2.1 Objective function

The objective function purpose is to minimize the latency. First, we assigned some delay to the links and put in the objective function.
Finally, we used the variable procdelay, which is another variable, but, in this case, of the node, so the function will minimize the sum of delay.

$$\sum_{(s,d)\epsilon L_{s,d}} (c_{p}^{s,d} * \lambda[s, d])$$

Illustration 11: Node delay

```
/* OBJECTIVE FUNCTION */

minimize COST: sum{(i,j) in L} ( y[i,j] * ( delay[i,j] ) ;
```

Illustration 12: Objective function

3.2.2 Constraints

There are a few constraints in the software.

- Flow conservation: it is a constraint that we have in all the programs, it is necessary. It says that the flow in the source is the same as in the destination node.

-Capacity constraint: we assign bandwidth to all the nodes, so the demand can't be higher than the capacity of the node.

$$\sum_{(s,d)\epsilon L_{s,d}} *f[i,j,k] * i_s^k = \sum_{(s,d)\epsilon L_{s,d}} *f[i,j,k] * i_d^k$$

Illustration 13: Flow conservation

$$\sum_{(k)\epsilon D_{s,d}^k} *f[i,j,k] * k <= L_{s,d} * B_{s,d}^v$$

Illustration 14: Capacity constraint

```
s.t. FLOW_CONSERVATION{i in N, k in dd}:
   sum{(j,i) in L} f[j,i,k] + (if i = source[k] then 1) =
   sum{(i,j) in L} f[i,j,k] +
   (if i = destination[k] then 1);

/* Capacity constraint */
s.t. CAPACITY{(i,j) in L}:
    sum{k in dd}f[i,j,k]*demand[k] <= y[i,j]*capacity[i,j] ;
```

Illustration 15: Capacity constraint first program

In illustration 16, there is the creation of the nodes. At this moment, with parameter procdelay; in the next programs, we will put more characteristics.

```
/* Node parameters, now we have the number of the node */

#number procdelay
alpha   2

bravo   2

charlie 2

delta   2

echo    2

foxtrot 2

golf    2
;

param: dd:  source  destination     demand  dweight:=

/* Demand parameters */

1   alpha   golf    100 45
;
```

Illustration 16: Node and path creation

Finally we created links, the path that we want to find/optimize, and the connections of the network.

```
param: L:   capacity    weight  delay:=

/* In links we have source destination bandwith and weight */

#s      d       bw      w   dl
alpha   bravo   200     1   1

alpha   charlie 200     5   2

alpha   delta   200     6   2

bravo   golf    200     1   1

charlie echo    200     3   7

delta   foxtrot 200     6   2

echo    golf    200     8   2

foxtrot golf    200     7   2
;


end ;
```

Illustration 17: Network creation with capacity weight and delay

We continued developing the program, and added the process delay, which simulates the time that is processing the node when the flow is crossing the node.
The remaining part is the same as the previous program, which will be called “minimize LATENCY”, as in the first version.

$$\sum_{(s,d)\epsilon L_{s,d}} \left(c_p^{s,d} * \left(\lambda_{s,d}^{v} + \lambda_{s,d}\right)\right)$$

Illustration 18: Node process delay

```
/* OBJECTIVE FUNCTION */

minimize LATENCY: sum{(i,j) in L} ( y[i,j] * ( delay[i,j] + procdelay
[i] ) ) ;

s.t. FLOW_CONSERVATION{i in N, k in dd}:
   sum{(j,i) in L} f[j,i,k] + (if i = source[k] then 1) =
   sum{(i,j) in L} f[i,j,k] +
   (if i = destination[k] then 1);

/* Capacity constraint */
s.t. CAPACITY{(i,j) in L}:
    sum{k in dd}f[i,j,k]*demand[k] <= y[i,j]*capacity[i,j] ;
```

Illustration 19: Objective function with process delay

3.2.3 NFV constraint

The next version that we did included a new constraint called PASS_THROUGH_NF.

With this constraint, we force the flow to go through the node that is doing the functions of Network Function Virtualization.

```
/* NFV constraint pass */
s.t. PASS_THROUGH_NF{k in dd}:
   sum{(i,j) in L} f[i,j,k]*vfn[i,j] >= 1;
```

Illustration 20: NFV constraint

In this version, we increased the network, because we have more advantages. For example, there are hundreds of new paths.

With the new network, we added more NFV and CONTROL nodes, which implies more possibilities to study what is going to happen, but the problem is that the we had to add 250 lines to our code, and became more complicated to manage.

```
/* Node parameters, now we have the number of the node */
#number procdelay   cost
alpha   1           300
bravo   1           300
charlie 1           300
delta   1           300
NFV     1           1500
foxtrot 1           300
CONTROL 1           1000
hotel   1           300
india   1           300
juliett 1           300
kilo    1           300
lima    1           300
mike    1           300
november    1       300
oscar   1           300
papa    1           300
CONTROL2    1       1000
romeo   1           300
NFV2    1           1500
tango   1           300
uniform 1           300
victor  1           300
;
```

Illustration 21: New program nodes.

3.3 Files and script

In the next version of the program, we solve that problem. We made the program to read the data from another file so the main program only has 130 lines.

```
/* reading data from table*/
table demand_data_table IN "CSV" "data_demand.csv":
dd <- [DEMAND_NUMBER], source~source, destination~destination,
demand~demand, dweight~dweight;
```

Illustration 22: Reading data from another file

This option we have done from nodes, links and the paths that we want.

Illustration 23: Files that the main program has to load

The files have the extension .csv, as we can see in the Illustration 19. We have to be very careful with the spaces at the end of the line, because it is really easy to make a mistake that will make the program to not compile.

```
NODE_NAME,procdelay,costN,type
alpha,1,300,1
bravo,1,300,1
charlie,1,300,1
delta,1,300,1
NFV,5,1500,2
foxtrot,1,300,1
CONTROL,1,1000,3
hotel,1,300,1
india,1,300,1
juliett,1,300,1
kilo,1,300,1
lima,1,300,1
oscar,1,300,1
mike,1,300,1
november,1,300,1
papa,1,300,1
rol22,1,300,1
romeo,1,300,1
rol2,1,300,1
tango,1,300,1
uniform,1,300,1
victor,1,300,1
```

Illustration 24: Example of csv file

Another improvement we made was to create as script in Linux which allowed us to run

the program writing ./script.sh in the console.
The script is simple, but really useful when you have to debug the program and run the GLPK-GNU command.

```
#!/bin/bash
rm camino.csv
clear
echo "Compiling files.."
echo "Executing gmpl.."
glpsol --model v0.12Al.mod -o solMan.txt
echo "Done!"
```

Illustration 25: Commands of the script

The “gplsol” is the command in the GLPK-GNU package. This command solves the file v0.12AL.mod and writes the output in solMan.txt.

We designed the network in R using Rstudio because, if we have an image of it, it is easier to understand what is happening, checking if the paths that the program has selected are correct using the constraints that we have programmed.

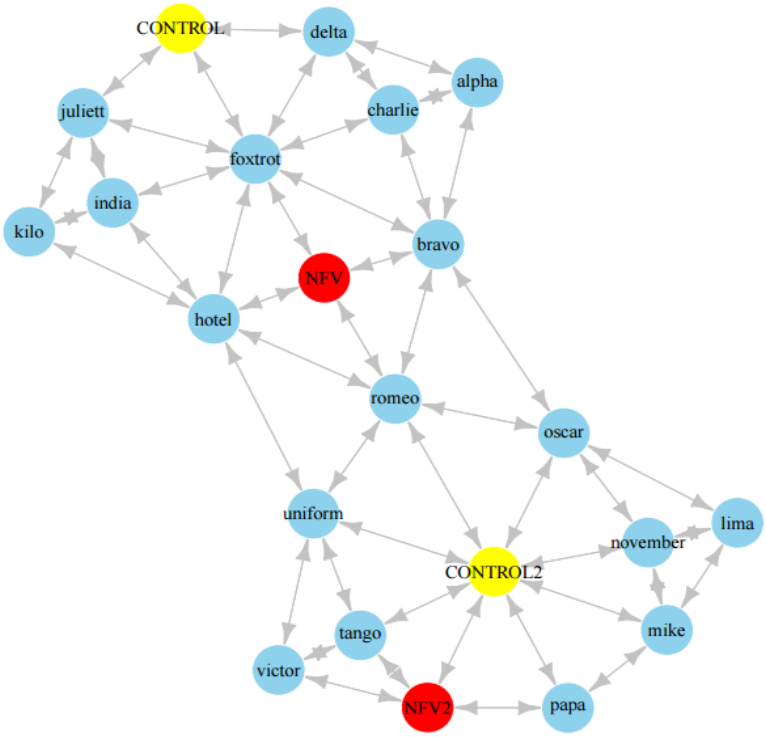

Illustration 26: Network design

3.4 Heuristic Programming

3.4.1 Introduction

Heuristic is a technique designed to solve problems quickly when classic methods are too slow, or to find an approximate solution when classic methods fail to find any exact solution.
An heuristic function, also called simply an heuristic, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow, it may approximate the exact solution.

3.4.2 Objective

Our objective is to use heuristic programming to solve our path problem faster. We wrote the code in python, and run it in Jupyter.

Finally, we will do some comparatives between linear programming and Heuristic to see the results.

3.4.3 Functions

First, we load some libraries to plot several graphs using Python.

```
import pandas as pd
import networkx as nx
import util as u
import random
```

Illustration 27: Python libraries

Then, we load the files, generate the graph, and run the algorithm to calculate what we want.

```
#NODE_NAME,procdelay,costN
NODE_INFO = pd.read_table('data_nodes.csv', sep=',',  engine='python',)
#FROM,TO,capacity,weight,delay,bwcontrol,vfn
NODE_LINKS= pd.read_table('data_links.csv', sep=',', engine='python',)
DEMAND = pd.read_table('data_demand.csv',sep=',', engine='python',)
#ata_links = pd.merge(NODE_ORIGIN, NODE_DESTINY,NODE_CAPACITY,NODE_WEIGHT,NODE_DELAY,NODE_BWCONTROL,NODE_ISNFV)
#data_nodes = pd.merge(NODE_NAME.to_frame(),NODE_DELAY.to_frame(),NODE_COST.to_frame())
G = nx.Graph()
for i in NODE_INFO.index.values:
    G.add_node(NODE_INFO['NODE_NAME'][i],procdelay=NODE_INFO['procdelay'][i],costN=NODE_INFO['costN'][i],capacity=NODE_INF
for i in NODE_LINKS.index.values:
    G.add_edge(NODE_LINKS['FROM'][i],NODE_LINKS['TO'][i],capacity=NODE_LINKS['capacity'][i] ,weight=NODE_LINKS['weight'][i
```

Illustration 28: Generation of the graph

The used algorithm is A*, with Python's implementation. The explanation is in annex.

3.5 Comparative Linear Programming vs Heuristic with python

3.5.1 CPU Comparative

The first comparative will be the CPU usage. We detected that the management is the same, when we run linear programming and heuristic with Python. Both types put the CPU usage at 99% when trying to solve the problem.

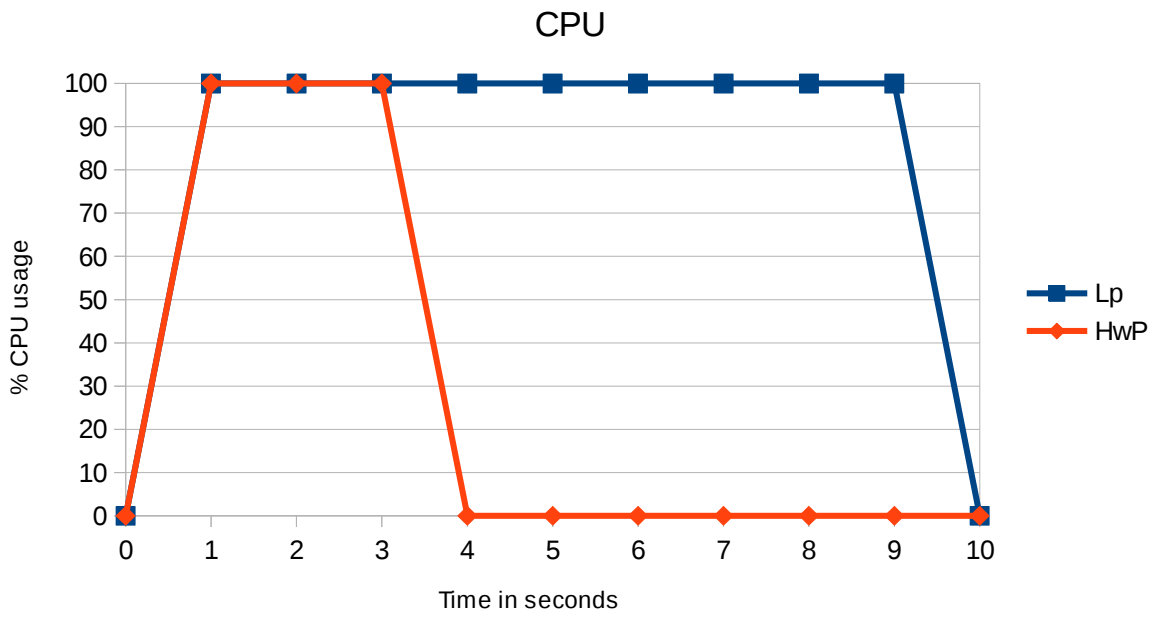

Illustration 29: CPU usage chart

In Illustration 29, we show 10 seconds as an example. During the rest of the program, it remains the same. As we can see in the image, it has some delay until it gets to 100%. The same happens at the end.

3.5.2 Time path calculation

Now we are going to show the comparative between paths.
In illustration 30, we can see the network that we are going to load. The image is generated with Rstudio, and it has 2 Control nodes and 2 NFV nodes. The constrains are the same as we showed previously.

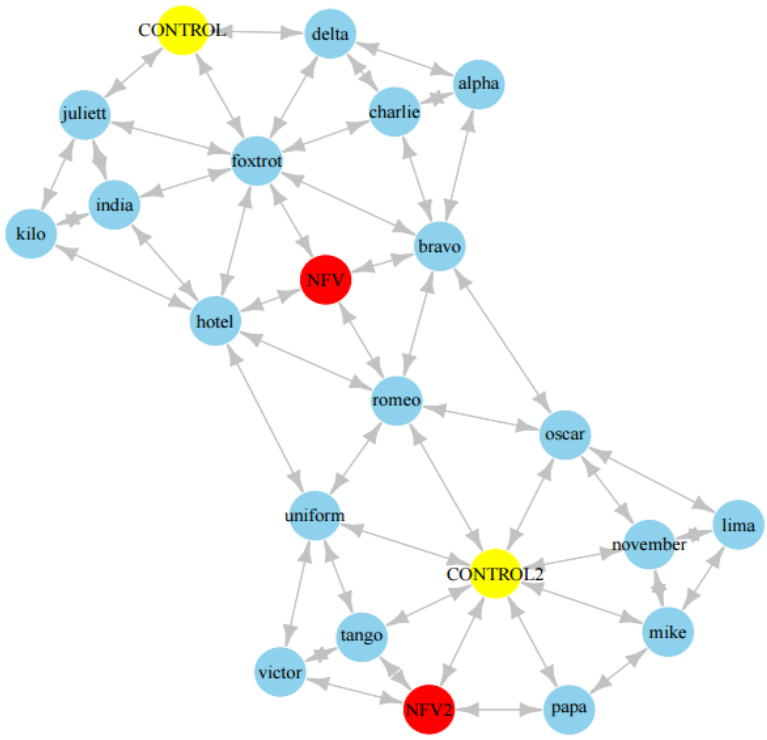

Illustration 30: Initial network

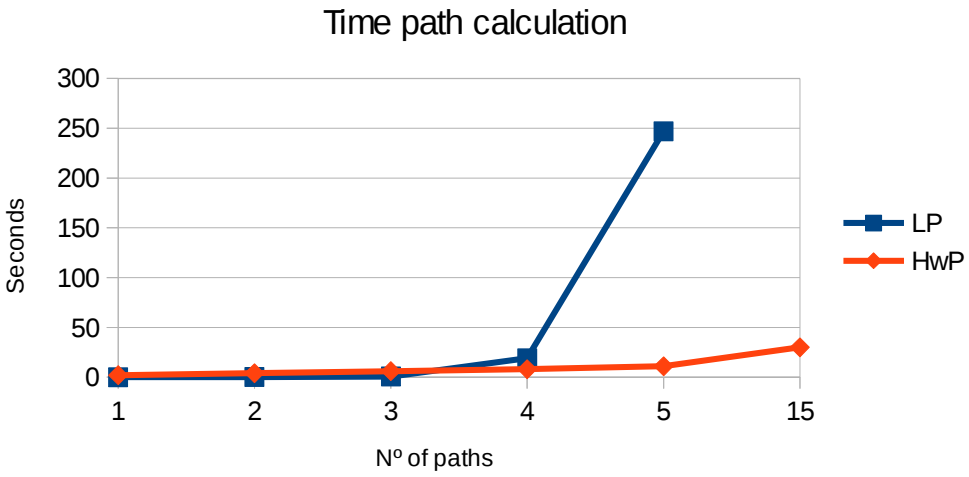

*Illustration 31: Time path calculation with Initial network**

*There is a special number in x-axis Nº of paths in where we pass from 7 to 15; we do that to prove that HwP is much faster with triple of nodes.

In illustration 31, we can see time path calculation in file demand; we added paths that the program has to solve.
First, we did with 1 path, then 2 paths, and 3 paths, and the result was almost identical.
When we have 4 paths, LP adds some delay, even if it could be negligible. But, when we add 5 paths, the difference is considerable, with almost 2 minutes of waiting compared to the 10 seconds using python.

3.5.3 Network reduced

In this simulation, we reduced the network. Now we have 1 Control and 1 NFV. The number of nodes is 11, instead of the 22 we had before.
Compared to illustration 31, now, when we have 5 paths, the time is almost the same, 10 seconds, but the same issue happens at step 6.
We can see that, using heuristic programming with python, the times remain the same, even if we increase the number of paths.

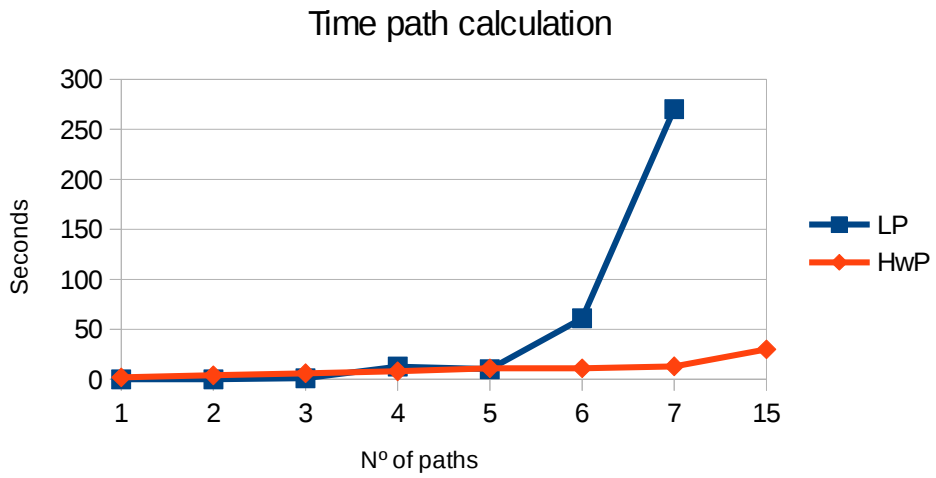

*Illustration 32: Time path calculation with network reduced**

* There is a special number in x-axis Nº of paths in where we pass from 7 to 15; we do that to prove that HwP is much faster with double of nodes.

3.5.4 Delay path calculation

Here we can find the delay path calculation.
There are two parameters that we are minimizing: delay of the link (Illustration 12), and process delay of the machine, a new variable we had to add. The simulation consists in finding the best path of every demand we ask for.

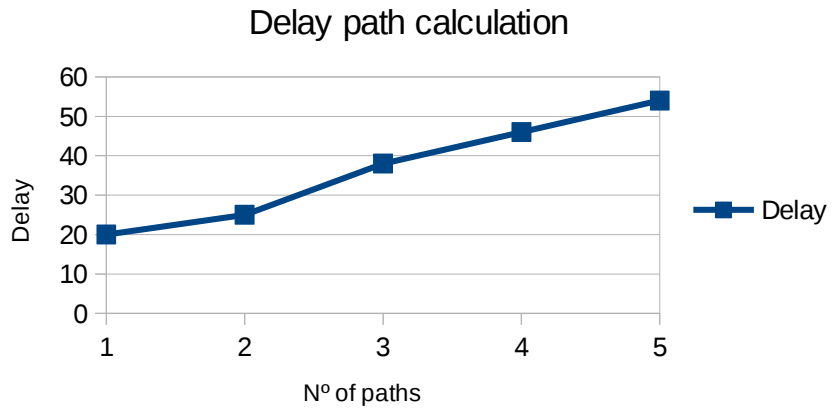

Illustration 33: Delay path calculation

3.5.5 Energy saving calculation

This illustration is minimizing the nodes that we are going to use to go through the entire path. For example, if node “Hotel” is used in 1st path, LP is going to try to use that node to reach the destination in the following paths.

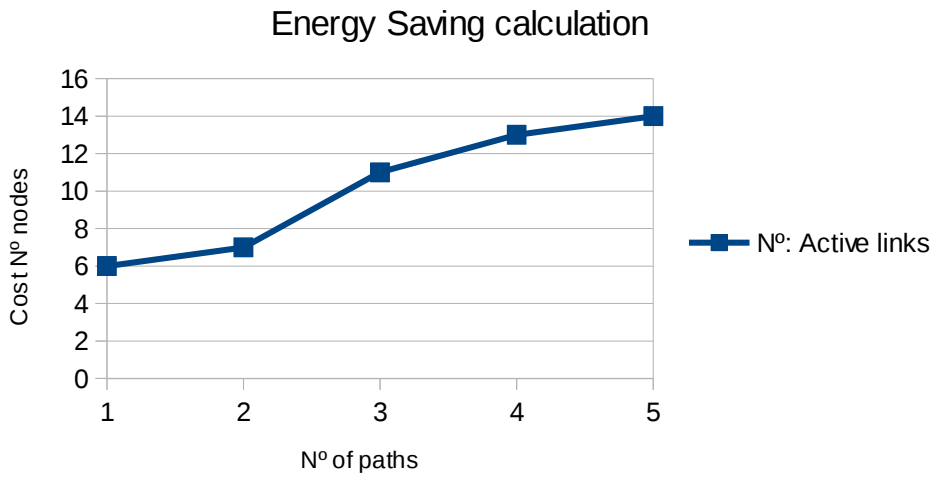

Illustration 34: Energy saving calculation

In this graphic, we can see that from 1 to 3 paths, the number of visited nodes doubled; but if paths go from 3 to 5, it only increases by 3 nodes. That is because the algorithm is using the same nodes every time it is possible, even if it is not the optimum path.

3.5.6 Random network

With the previous illustrations, we could see how powerful simulation with Python is. With our 22 node network, the computer solves the network optimization easily.
Now, we want to find how powerful the computer that we are using is doing simulations with massive networks.

With this command, python creates random network:

```
Grand=nx.dense_gnm_random_graph(20000,50000)
```

Illustration 35: Python command for random graph.

This network is one simulated in illustration, with 20k nodes and 50k links. The computer takes less than 20 seconds to simulate that.

In illustration 36, we can see a piece of code where python's timer is used to calculate the time.
There is also a loop to calculate the network 25 times in order to get the best time.

```
start = timeit.default_timer()

for i in range (25):
    nx.astar_path(Grand, random.randint(0,9999),random.randint(0,9999),heuristic=simpleHeuristic, weight='weight')

stop = timeit.default_timer()
```

Illustration 36: Python code for calculate the solution.

Here we have all simulations to find the limit of our computer.

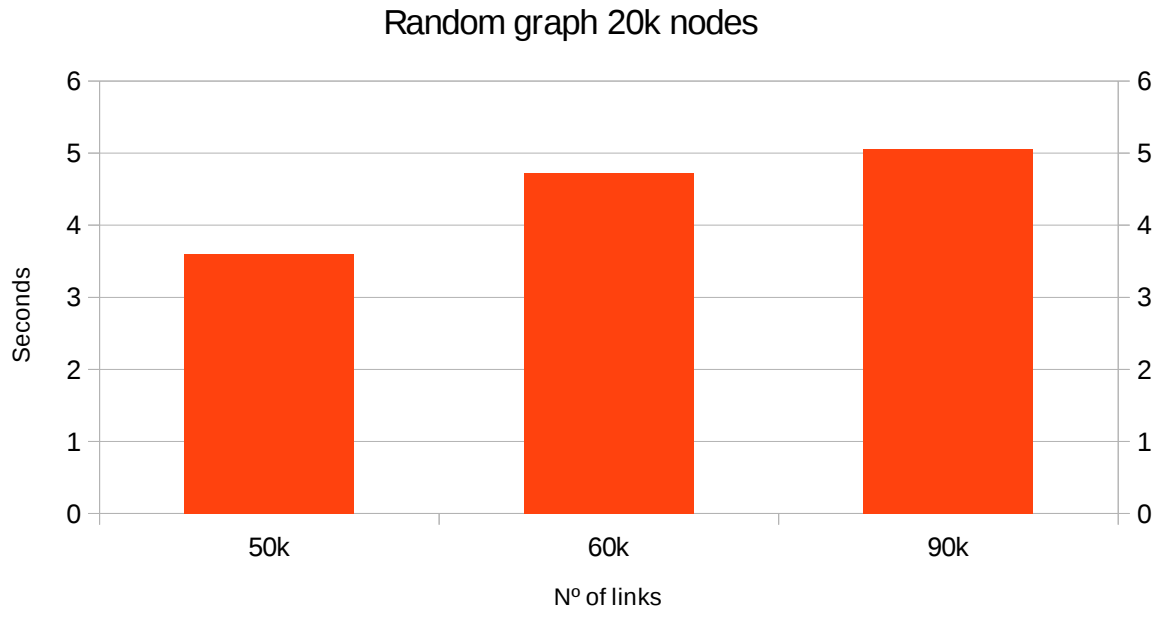

Illustration 37: Python simulation of 20k nodes

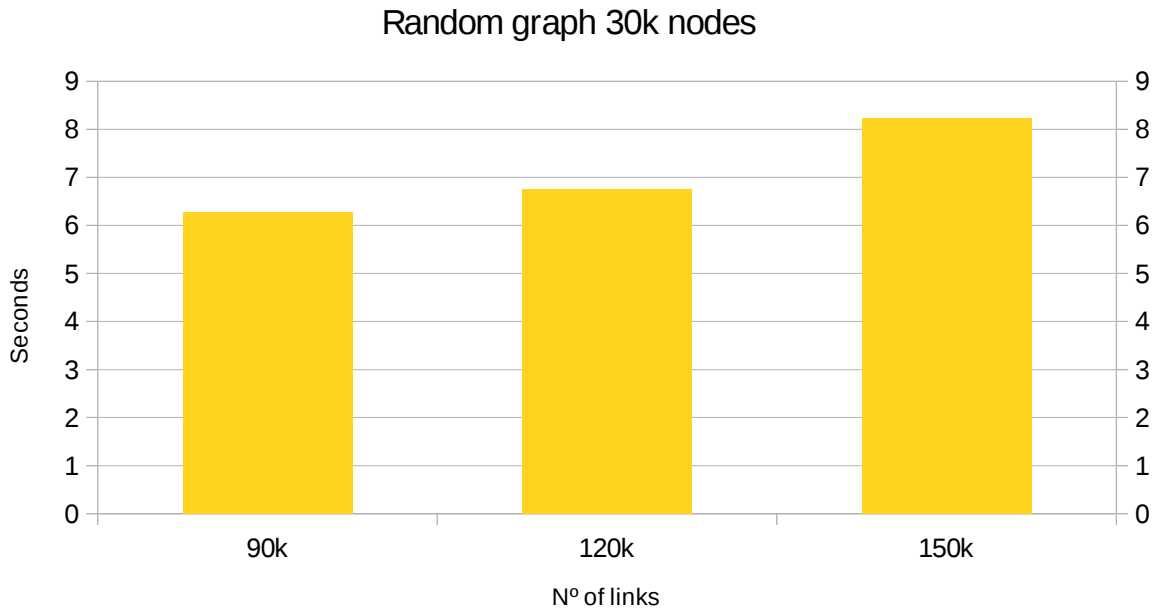

Illustration 38: Python simulation of 30k nodes

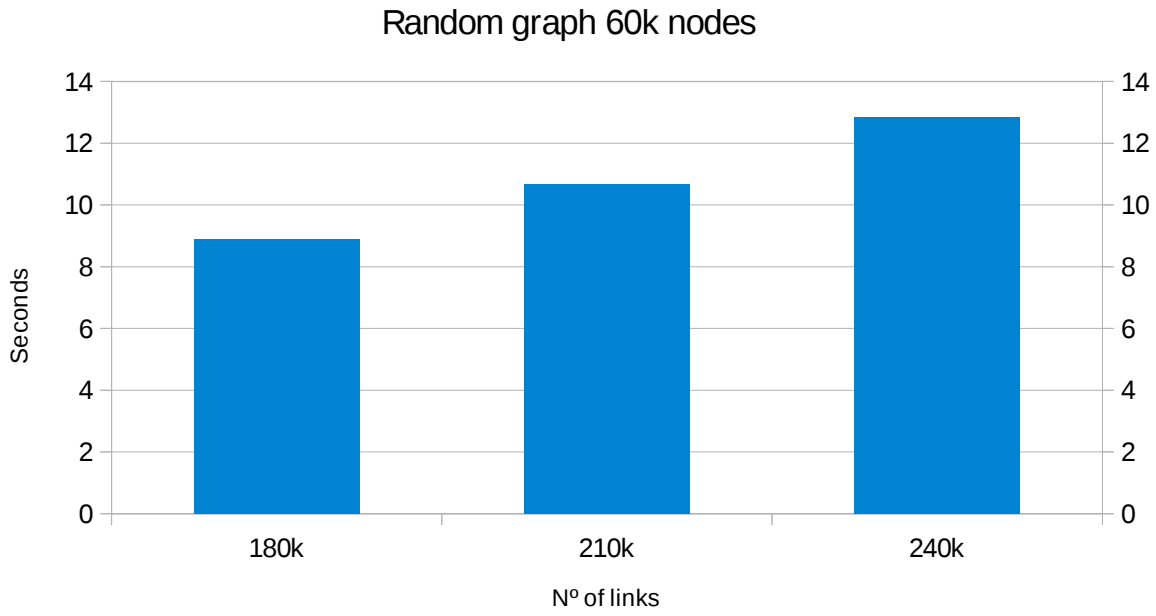

Illustration 39: Python simulation of 60k nodes

3.5.7 Generic constrained path-finding algorithm

Although the LP model allows the attainment of optimal solutions for the constrained short-path problem in SDN/NFV, it becomes challenging to solve on large and even medium-scale topologies. This is because the difficulty of this routing problem is known to be NP-Hard, so the consumption of resources and time complexity grow exponentially with the network size. To reduce these metrics, in this section we develop a heuristic algorithm.

We consider random networks of 100 nodes, with different edge rewiring probability, p. Nodes in sparse networks have p=0.1; in medium networks p=0.25, and in dense networks between p=0.6. For each category we generated 100 networks.

In this case, we have determined the controllers position solving the optimal controller placement problem as a clustering analysis problem and using the k-means clustering algorithm to find the global optimal. We have also considered the same criterion for deciding the NFV nodes position. The required number of both kind of nodes has been fixed to 5 for sparse networks, 3 for medium networks and 1 for dense networks. In all cases, this is the maximum number of controllers for which adding another controller results in worsening the acceptance ratio (due to data path do not cross through controllers).

The following figure shows a comparison of the acceptance ratio of demands in case of having the NFV constraints (i.e., find paths that pass trough at least one NFV node) and in case of having just shortest paths without this constraint. In both cases, paths do not cross controllers nodes.

The objective of this figure is to analyse the influence of the NFV constraint in the pathfinding procedure. It can be seen that the difference between both methods is less than 5% in the worst case of our analyses (when the acceptance ratio decreases to 85%).

We present the simulation results for the heuristic together with their respective 95% confidence intervals based on a standard normal distribution. With this heuristic algorithm we have a tool to design and deeply analyse the SDN/DFV network but, it is beyond the scope of this project.

Graph of Acceptance ratio of demands

Net of 100 nodes with edge wiring probability 0.1

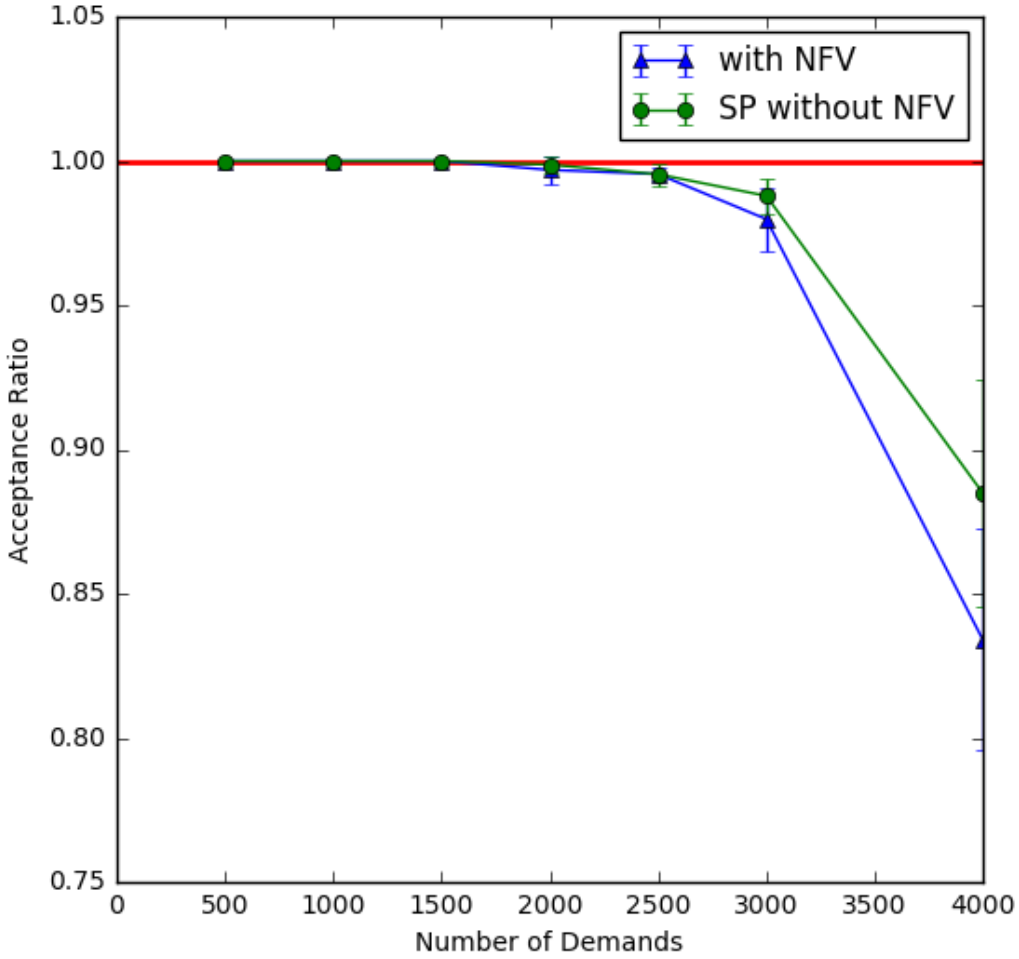

Illustration 40: Pathfinding comparison between heuristic and Short-Path models

CHAPTER 4 CONCLUSIONS

4.1 Conclusion

Once the project is over, we can see that our objectives are accomplished.
The main concepts, SDN and NFV, are explained in a clear and easy way, with some examples and illustrations.

We have seen two different programming languages: Linear Programming and Heuristic with Python. The comparative was clear: Python is better if there are more than 5 simultaneous paths, which it could happen easily in real life.

Another interesting Python characteristic are the simulations we did with random network. We had more than 20k-30k nodes with 80k links.

All the used software is free: the machine was running Debian 8.6, OpenOffice as text processor, and all the packets use: GLPK-GNU, Rstudio, Okular and Python.
The experience using free software was absolutely satisfying. Most of the times, you only needed to know what package you wanted to install and, with only a command, everything was fast and easily installed. Everybody can now see the results of this project.

During the realization of this project, we had some problems that were solved during the following weeks. Learning the GLPK-GNU language was challenging. It is difficult to find information and examples in internet and the learning process was laborious. With the other programming language, Python, it was more accessible and smooth.

Finally, I am absolutely satisfied with this project and I want to thank Cristina for her collaboration, ideas, advices, cooperation, and support I needed during this months.

4.2 Mention

[1]https://gigaom.com/2014/04/02/google-launches-andromeda-a-software-defined-network-underlying-its-cloud/

[2]http://www.networkworld.com/article/2937396/cloud-computing/microsoft-needs-sdn-for-azure-cloud.html

[3] https://es.wikipedia.org/wiki/Redes_definidas_por_software

[4] ttp://searchsdn.techtarget.com/tip/Centralized-vs-decentralized-SDN-architecture-Which-works-for-you

[5] https://www.sdxcentral.com/sdn/definitions/southbound-interface-api/

[6]ttp://searchsdn.techtarget.com/guides/Northbound-API-guide-The-rise-of-the-network-applications

[7]https://en.wikipedia.org/wiki/Network_function_virtualization

[8]https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/

[9]https://en.wikipedia.org/wiki/GNU_Linear_Programming_Kit

https://en.wikipedia.org/wiki/AMPL

https://en.wikibooks.org/wiki/GLPK

https://en.wikibooks.org/wiki/GLPK/GMPL_(MathProg)#Official_documentation

https://cran.r-project.org/web/packages/Rglpk/index.html

[10]https://www.r-project.org/

[11]https://cran.r-project.org/doc/contrib/Short-refcard.pdf

[12]https://en.wikipedia.org/wiki/A*_search_algorithm

4.4 Annex

4.4.1 GLPK - GNU Linear Programming Kit

The GNU Linear Programming Kit (GLPK) is a software package intended for solving large-scale linear programming, mixed integer programming, and other related problems.
It is a set of routines written in ANSI C and organized in the form of a callable library. The package is a part of the GNU Project and is released under the GNU General Public License.

Problems can be modelled in the language GNU MathProg which shares many parts of the syntax with AMPL and solved with standalone solver GLPSOL.

GLPK was developed by Andrew O. Makhorin (Андрей Олегович Махорин) of the Moscow Aviation Institute. The first public release was in October 2000.

-Version 1.1.1 contained a library for a revised primal and dual simplex algorithm.
-Version 2.0 introduced an implementation of the primal-dual interior point method.
-Version 2.2 added branch and bound solving of mixed integer problems.
-Version 2.4 added a first implementation of the GLPK/L modelling language.
-Version 4.0 replaced GLPK/L by the GNU MathProg modelling language, which is a subset of the AMPL modelling language.

Characteristics:

- Original author: Andrew O. Makhorin
- Developers: GNU Project
- Stable release: 4.58 / February 18, 2016
- Written in: C
- Operating system: Cross-platform
- License: GNU General Public License

¿What is AMPL?

A Mathematical Programming Language is an algebraic modelling language to describe and solve hight-complexity problems for large-scale mathematical computing.
It was developed by Robert Fourer, David Gay, and Brian kernighan at Bell Laboratories.
AMPL supports dozens of solvers, both open source and commercial software.
AMPL is used by more than 100 corporate clients, and by government agencies and academic institutions.

Characteristics:

Paradigm: multi-paradigm: declarative, imperative.
Designed by: Robert Fourer, David Gay, Brian Kerninghan
Developer: AMPL Optimization
First appeared: 1985, 31 years ago
Stable release: October 12, 2013
Operating System: Cross-platform.

Filename extensions: .mod

4.4.2 R

The R[10] software is a free software environment for statistical computing and graphics.
It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories by John Chamerts and colleagues. R can be considered as a different implementation of S.

R provides a wide variety of statistical (linear and non-linear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form.
It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

With the s.o that I am doing this work, (cat /etc/*-release)

PRETTY_NAME="Debian GNU/Linux 8 (jessie)"
NAME="Debian GNU/Linux"
VERSION_ID="8"
VERSION="8 (jessie)"
ID=debian
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

The program R is installed so we can start working with the software.
If you are working with windows or Mac, in the link:

https://cran.r-project.org/mirrors.html

You can find the mirror of your county and download the package for install the software.

When the R software is executed, it looks like a simple console which you can find in any operating system, but investigating I have found a software that complements R quite good.

4.3.3 RStudio

RStudio[11] is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

For install Rstudio you only have to download Rstudio Desktop and execute this command:

Download in:

https://www.rstudio.com/products/rstudio/

For install in linux (debian8):

sudo dpkg --install rstudio-0.99.902-amd64.deb

4.3.4 A* Algorithm

Is a computer algorith[12] that is generally used in pathfinding and graph traversable path between multiple points, called nodes. Noted for its performance and accuracy, it enjoys widespread use.

Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first described the algorithm in 1968. It is an extension of Edsger Dijkstra's 1959 algorithm.
A* achieves better performance by using heuristics to guide its search.
Solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution.

4.3.5 Illustrations

4.3.6 Python code of the algorithm

```
import matplotlib.pyplot as plt
from random import randint
import operator
import numpy as np
from numpy import *
from scipy.cluster.vq import vq, kmeans, whiten
import scipy.stats as stats
```
```python
# Define k Controller positions
# We solve the optimal controller placement problem as clustering analysis problem, and use the
# typical clustering algorithm named K-means algorithm to find the global optimal
# solution.
def DefineControllers(G, k):
controllers = []
# Convert adjacencies matrix into numpy matrix
A = nx.to_numpy_matrix(G)
whitened = whiten(A)
book = array((whitened[0],whitened[2]))
km=kmeans(whitened,book)
mat= km[0][0]
while len(controllers) < k:
c=mat.argmax()
if c not in controllers:
controllers.append(c)
mat = np.delete(mat, c)
return(controllers)
```
```python
# Define k NFV positions. We use the same criterion than in controllers location definition
def DefineNFV(G,listcontrollers, k):
nfv = []
# Convert adjacencies matrix into numpy matrix
A = nx.to_numpy_matrix(G)

whitened = whiten(A)
book = array((whitened[0],whitened[2]))
km=kmeans(whitened,book)
mat= km[0][0]
while len(nfv) < k:
c=mat.argmax()
if c not in listcontrollers:
if c not in nfv:
nfv.append(c)
mat = np.delete(mat, c)
return(nfv)
```

```
```python
# Choose a node in the graph G (nor NFV neither controller)
def ChooseNode(G):
n=randint(0,NumNodes-1)
while n not in G.nodes() or n in nfv:
n=randint(0,NumNodes-1)
return(n)
```
```python
# Determine demands randomly
def ObtainDemands(G, NumDemands):
drequest = {}
DEMAND = []
num = 0
while num < NumDemands:
drequest = {
'source': ChooseNode(G),
'destination': ChooseNode(G),
'demand': 100,
'dweight': 5,
}
if drequest['source']!= drequest['destination'] and \
(drequest['source'],drequest['destination']) not in G.edges():
DEMAND.append(drequest)

num = num+1
return DEMAND
```
```python
# Heuristic to be used in the A* ("A star") algorithm via
networkx.algorithms.shortest_paths.astar
# inside Acceptance function
def Heuristic(node1, node2):
global G2
fullcost = 0
if G2.get_edge_data(node1,node2) is None :
fullcost = 1
return fullcost
elif G2.get_edge_data(node1,node2)['vfn']:
return 1
else:
fullcost = G2.get_edge_data(node1,node2)['weight'] + G2.node[node1]['costN']+\
G2.node[node2]['costN']
return fullcost
```
```python
# Compute the acceptance ratio of demands
def Acceptance(G, controllers, NumDemands):
global G2
```
```

```
# For searching data paths SDN control nodes are removed
G2rand = G.copy()
for i in controllers:
G2rand.remove_node(i)
# Determine demands randomly
DEMAND = ObtainDemands(G2rand, NumDemands)
numsuccesful=0
numNONsuccesful=0
lengths=0

for i in range(NumDemands):
G2 = G.copy()
# Eliminate edges with no enough capacity for the current demand
for j in G2.edges():
if G2.get_edge_data(j[0],j[1])['capacity'] <= DEMAND[i]['demand']:
G2.remove_edge(j[0],j[1])
try:
m = nx.astar_path(G2, DEMAND[i]['source'],DEMAND[i]
['destination'],heuristic=Heuristic)
numsuccesful = numsuccesful+1
lengths=lengths+len(m)
# Reduce the link capacity due to current demand value
for k in range(len(m)-1):
G[m[k]][m[k+1]]['capacity'] = G[m[k]][m[k+1]]['capacity'] - DEMAND[k]['demand']
except nx.NetworkXNoPath:
numNONsuccesful=numNONsuccesful+1
ratio = numsuccesful/(numsuccesful+numNONsuccesful)
lmean = lengths/NumDemands
return(ratio,lmean)
```
```python
# Compute the acceptance ratio of demands in case of Shortest-Paths without the
constraint of passing through at
# least one NFV node
def AcceptanceSP(G, controllers, NumDemands):
# For searching data paths SDN control nodes are removed
G3rand = G.copy()
for i in controllers:
G3rand.remove_node(i)
# Determine demands randomly
DEMAND = ObtainDemands(G3rand,NumDemands)
numsuccesful=0
numNONsuccesful=0

lengths=0
for i in range(NumDemands):
G3 = G.copy()
# Eliminate edges with no enough capacity for the current demand
for j in G3.edges():
if G3.get_edge_data(j[0],j[1])['capacity'] <= DEMAND[i]['demand']:
```

```
G3.remove_edge(j[0],j[1])
try:
m = nx.shortest_path(G3,DEMAND[i]['source'],DEMAND[i]['destination'])
numsuccesful = numsuccesful+1
lengths=lengths+len(m)
# Reduce the link capacity due to current demand value
for k in range(len(m)-1):
G[m[k]][m[k+1]]['capacity'] = G[m[k]][m[k+1]]['capacity'] - DEMAND[k]['demand']
except nx.NetworkXNoPath:
numNONsuccesful=numNONsuccesful+1
ratio = numsuccesful/(numsuccesful+numNONsuccesful)
lmean = lengths/NumDemands
return(ratio,lmean)
```

```python
def GenerateGraph(NumNodes, p, NumControllers, NumNFV):
Grand=nx.erdos_renyi_graph(NumNodes, p, seed=None, directed=False)
nx.set_edge_attributes(Grand, 'capacity', 2000)
nx.set_edge_attributes(Grand, 'weight', 8000)
nx.set_edge_attributes(Grand, 'delay', 1)
nx.set_edge_attributes(Grand, 'bwcontrol', 1)
nx.set_edge_attributes(Grand, 'vfn', 0)
nx.set_node_attributes(Grand, 'costN', 300)
nx.set_node_attributes(Grand, 'capacity', 300)
nx.set_node_attributes(Grand, 'procdelay', 1)
controllers = DefineControllers(Grand,NumControllers)
nfv = DefineNFV(Grand,controllers,NumNFV)
for i in controllers:
Grand.node[i]['name'] = 'CONTROL'+str(controllers.index(i)+1)

Grand.node[i]['costN'] = 1000
for i in nfv:
Grand.node[i]['name'] = 'NFV'+str(nfv.index(i)+1)
Grand.node[i]['procdelay'] = 5
Grand.node[i]['costN'] = 1500
# Change the value of vnf field in links connected to NFV nodes
li=[]
for i in range(Grand.number_of_nodes()):
for n in Grand.edge[i].keys():
if n in controllers:
continue
if n in nfv:
if i not in li: li.append((n,i))
for n,i in li:
Grand[n][i]['vfn'] = 1
Grand[n][i]['delay'] = 500
Grand[n][i]['weight'] = 1
return(Grand, controllers, nfv)
```

```python
```

```
NumNodes = 100
p = 0.1 # edge rewiring probability
NumControllers = 5
NumNFV = 5
ListDemands = [500,1000,1500,2000,2500,3000, 4000, 5000]
ListMeanAcceptance = []
ListMeanSPAcceptance = []
ListStdAcceptance = []
ListStdSPAcceptance = []
for NumDemands in ListDemands:
ListAcceptance = []
ListSPAcceptance = []
for i in range(100): #100 iterations for each case

print(NumDemands)
Grand, controllers, nfv = GenerateGraph(NumNodes, p, NumControllers,NumNFV)
# Compute the acceptance ratio of demands with the NFV contraint
ratio1, len1 = Acceptance(Grand, controllers, NumDemands)
ListAcceptance.append(ratio1)
Grand, controllers, nfv = GenerateGraph(NumNodes,p,NumControllers,NumNFV)
# Compute the acceptance ratio of demands
ratio2, len2 = AcceptanceSP(Grand, controllers, NumDemands)
ListSPAcceptance.append(ratio2)
ListMeanAcceptance.append(np.mean(ListAcceptance))
ListStdAcceptance.append(np.std(ListAcceptance))
ListMeanSPAcceptance.append(np.mean(ListSPAcceptance))
ListStdSPAcceptance.append(np.std(ListSPAcceptance))
```

```python
print(ListMeanAcceptance)
print(ListStdAcceptance)
```

```python
print(ListMeanSPAcceptance)
print(ListStdSPAcceptance)
```

```python
# Compute the confidence intervals
z_critical = stats.norm.ppf(q = 0.975) # Get the z-critical value
sample_size = len(ListStdAcceptance)
intervals1 = []
for i in range(sample_size):
margin_of_error = z_critical * (ListStdAcceptance[i]/math.sqrt(sample_size))
confidence_interval=(ListMeanAcceptance[i] - margin_of_error, \
ListMeanAcceptance[i] + margin_of_error)

intervals1.append(confidence_interval)
sample_size = len(ListStdSPAcceptance)
intervals2 = []
for i in range(sample_size):
```

```
margin_of_error = z_critical * (ListStdSPAcceptance[i]/math.sqrt(sample_size))
confidence_interval=(ListMeanSPAcceptance[i] - margin_of_error, \
ListMeanSPAcceptance[i] + margin_of_error)
intervals2.append(confidence_interval)
```
```python
# Plot the results
fig = plt.figure(figsize=(6,6))
x = np.array([500,1000,1500,2000,2500,3000, 4000])
y = np.array(ListMeanAcceptance)
e = np.array([(top-bot)/2 for top,bot in intervals1])
l1=plt.errorbar(x, y, e, marker='^', label='cas11')
y = np.array(ListMeanSPAcceptance)
e = np.array([(top-bot)/2 for top,bot in intervals2])
l2=plt.errorbar(x, y, e, marker='o', label='cas2')
plt.legend([l1, l2], ['with NFV', 'SP without NFV'])
plt.hlines(xmin=0, xmax=4000,
y=1,
linewidth=2.0,
color="red")
fig.suptitle('Acceptance ratio of demands\n Nets of 100 nodes with edge rewiring
probability 0.1', fontsize=12)
plt.xlabel('Number of Demands', fontsize=10)
plt.ylabel('Acceptance Ratio', fontsize=10)

fig.savefig('acceptance01.png')
plt.show()
```
```python
```