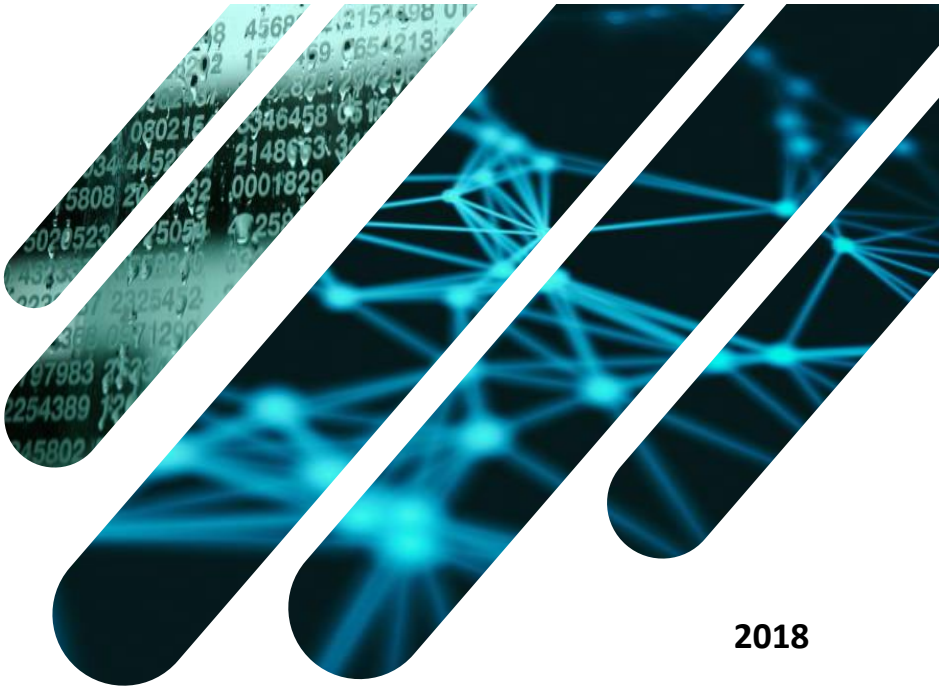
Nicolas Cassajus

Jonathan Malique



2018

Network building and path determination

Internship project documentation

F'SATI
French South African Institute of Technology

Mr Angus Brandt

SUMMARY

INTRODUCTION

Currently in fourth year of an engineering school in France, we went to F'SATI for our 3 months summer internship. Our project was about network building and path determination. First, we had to set up the environment on a server. Then, we could configure, build our topology, and retrieve it with REST requests, to display it and to visualize the network.

Finally, we could begin mathematics works on the network, algorithms as the k-shortest path, before using path optimization with minimization. This report is about the steps of our project and about helping to continue our project with some documentation. Here is a diagram of the full project with the different steps we managed to implement :

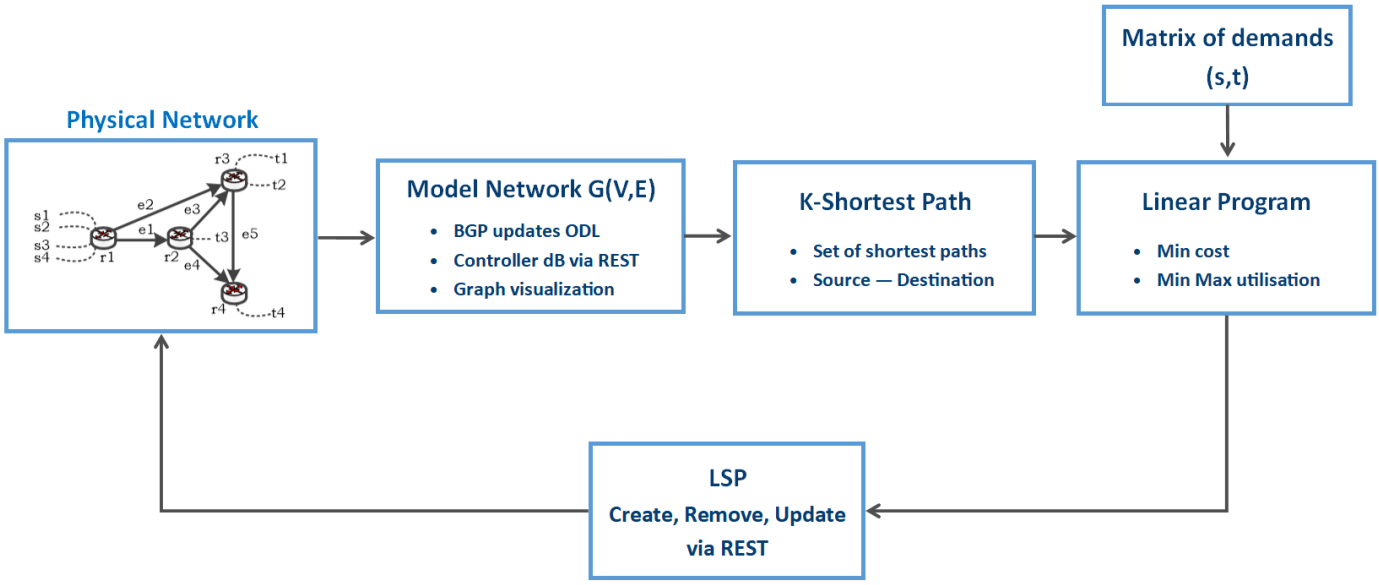


Figure 1 Project cycle diagram

I. Environment setting up

This is the first step of our project, we had a server, a screen and a CD of the latest release of Ubuntu (16.04 LTS). We had to install it on the server. After one week of installation, we managed to run it, there is only a problem of display for the login menu of Ubuntu, where everything is unreadable and striped.

To log in when you are in this window, just press ENTER after turning on the computer, then type "admin" and ENTER again. You should be log in, now.

Then, we installed Eclipse and VMWare for the next steps.

II. Network building

A. VMWare and the routers

- After running the environment, we had to build our network on it. First, VMWare is a software that allows to handle virtual machines. In fact, each router that we will create will be a virtual machine. For this, we use the Cisco IOS XR routers that are virtual routers. Here is a schema of the system operation of a router [1] :

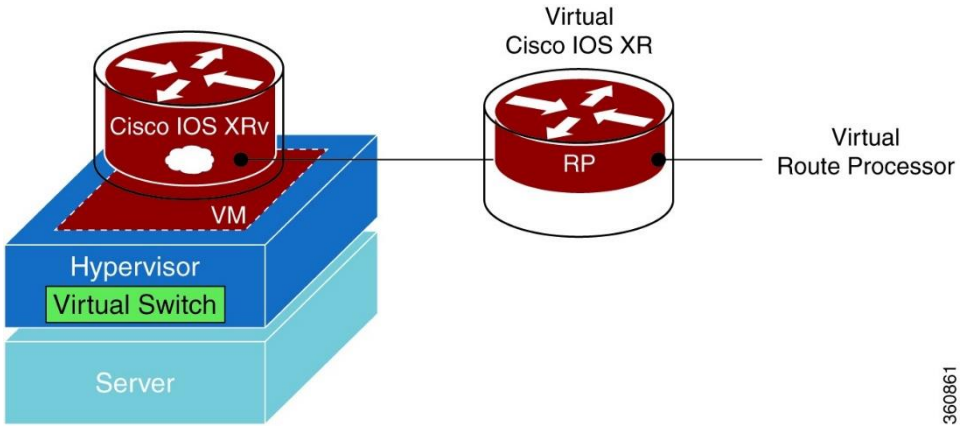


Figure 2 Cisco router on a VM

- When the computer is turning on, VMWare must be launched. Then, you can see a list of every router of the topology (xrv1, 2xrv3, …). Each router has to be started to build the network.

If you go in the settings of each router (right click on it), you can find the memory of a router, and all the interfaces that it has with every corresponding MAC address.

One of the interfaces is the Management interface, a network interface dedicated to configuration and management operations. You also have the virtual network interface. The other interfaces are used to connect the routers in the network.

[1] https://www.cisco.com/en/US/docs/ios_xr_sw/ios_xrv/install_config/b_xrvr_432_chapter_01.html

- If you need to add an interface for other routers and links, you will have to go to the configuration file situated at **/home/Documents/xrvs/[name of the router]/[name of the router].vmx** and to modify the vmx file to add a line for the new ethernet interface : **ethernet[n°].virtualDev = "e1000"**.

B. Configuring the routers (protocols)

- Once the routers are running, you can modify their configuration. For this you had to launch a terminal to execute a linux command : **sudo socat unix-connect:/tmp/[name of the router] stdio,raw,echo=0,escape=0x1a**

Once you had executed this line, you must log in with login "cisco" and password "cisco". You are finally in the configuration of the router that you can display with **show running-configuration**.

You can find in **Annex A** an example of configuration for the third router of the third Autonomous System.

C. Explanations about the configuration and the protocols

- We have used several protocols for each router to build our network as you can see in **Annex A**

In order to make our routers exchange, we used three main protocols: OSPF (Open Shortest Path First), RSVP (ReSerVation Protocol) and MPLS (Multi-Protocol Label Switching). OSPF is used to assign costs to each link in our topology, while RSVP assigns a given bandwidth. MPLS, is used to combine the two and create a routing that uses information from OSPF and RSVP instead of using classic routing rules.

For OSPF, a per area database system is created. That means that each router in each area has the same routing information and database.

RSVP by itself is not a routing protocol but is used to make reservations along a routing path to establish a definite QoS (quality of service)

Finally, MPLS is a routing protocol of layer 2.5 (between the OSI layer 2 and 3) and is centred around label switching, by opposition with the concept of packet switching that is often used in routing.

The information given by OSPF and MPLS are able to be retrieved by OpenDaylight, and as such, they can be used later by our Java program to calculate the most optimised path.

D. Opendaylight Controller

- The next step of our project was about configuring the controller on our network. We used the Opendaylight Controller, an open-source project, and the release Boron SR4 (0.5.4). We tried some new releases, but it did not work.

● At first, we had to build 3 different AS as you can see in **Figure 5** on the page 5. The goal was about create each AS on one server with one controller (Figure 3).

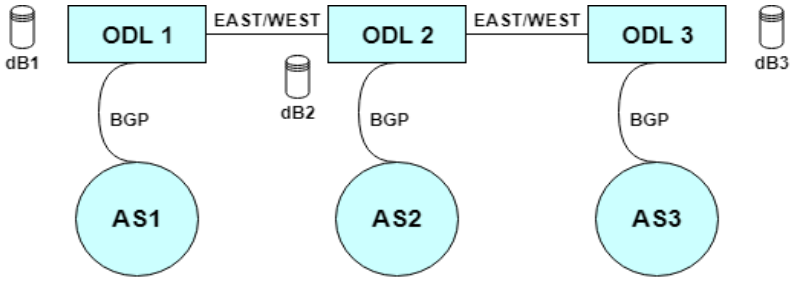


Figure 3 First goal of the project

However, we had a lot of troubles with the East/West connexion and we never managed to make it run. Then, we have used an easier way to move on in our project (Figure 4), by using one server and one controller, connecting the 3 AS to this controller.

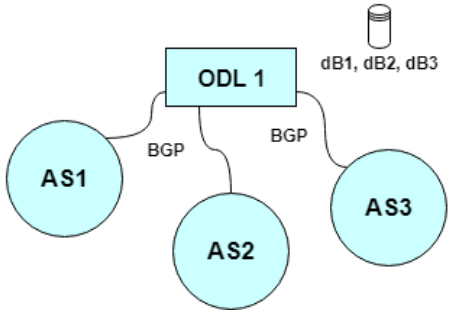


Figure 4 Other controller building (easier)

● To use the Opendaylight controller we first had to install the features on it. For this, we had to go to **/home/Documents/odl/distribution-karaf-0.5.4-Boron-SR4/bin/** then we launched **sudo ./karaf** which is a console allowing to install the features for Opendaylight. The necessary features were odl-bgpcep-pcep, odl-bgpcep-bgp, odl-dlux-core, odl-dlux-all, odl-restconf-all.

Once the installation of the necessary features was done, we had to go to the folder **/home/Documents/odl/distribution-karaf-0.5.4-Boron-SR4/etc/Opendaylight/karaf/** to modify the file "41-bgp-example.xml" to configure the controller (with the different addresses of our network and with the 3 topologies that it had to create).

Finally, to run Opendaylight, it is necessary to go back to **/home/Documents/odl/distribution-karaf-0.5.4-Boron-SR4/bin/** and to launch **sudo ./start**. You can verify if the controller is running by launching **sudo ./status** and you can stop it by using **sudo ./stop**.

Now, it is possible to access to Opendaylight via your internet browser and the address : **localhost:8181/index.html** (admin/admin).

Here, it is possible to go to Yang UI which allows to make some HTTP requests, to retrieve the topology for example (network-topology rev / operational / network-topology / Send / Display Topology).

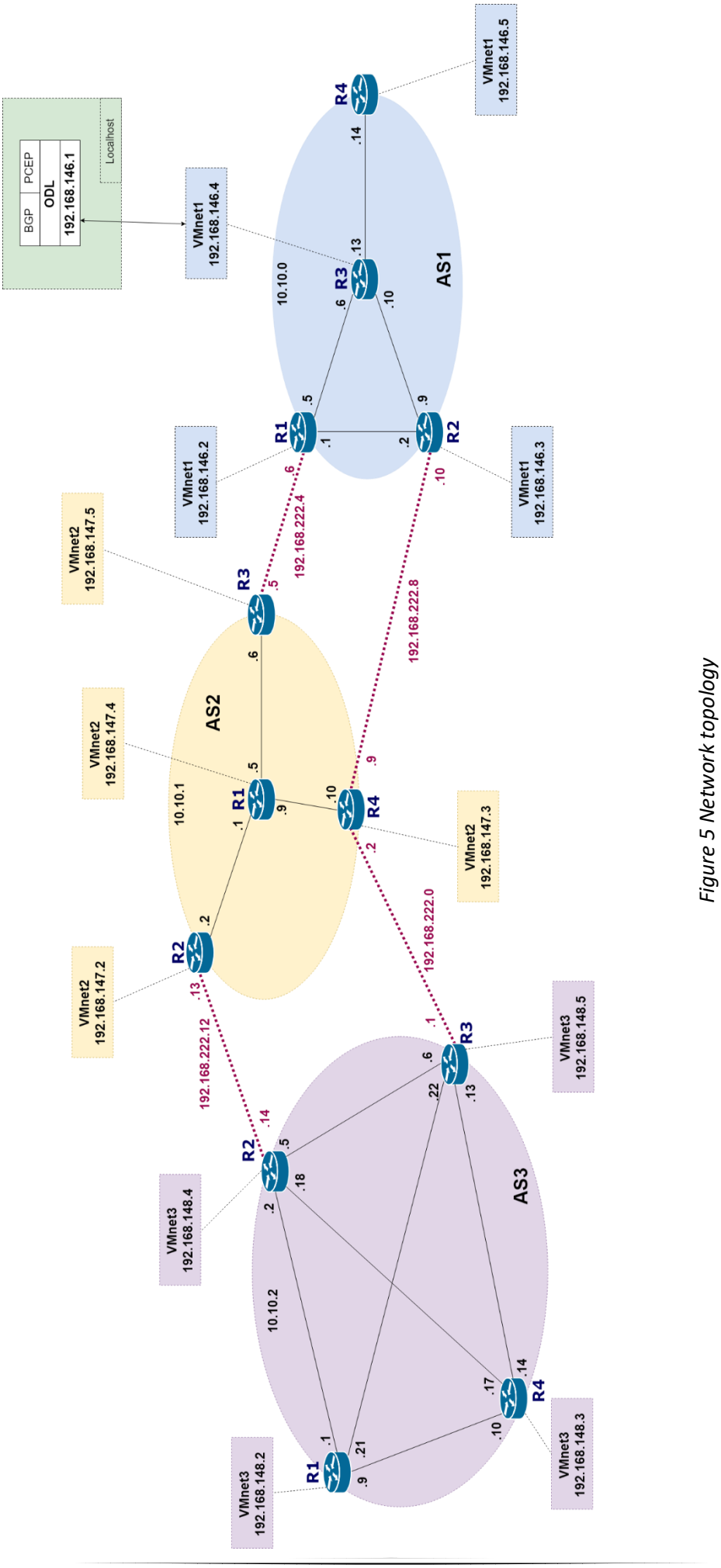


Figure 5 Network topology

III. Network visualization

After building our network and making the controller run, we had to visualize it. We used Pathman, an application made with NeXt UI. To run it, it is necessary to go to the folder **/home/Visualize/Pathmanbis/pathman/**.

Here are different Python files :

- **pathman_ini.py** : to define the address of the ODL controller, the login and password
- **pathman50.py** : to define how to build the graph with the topology
- **topology.py** : to define how to retrieve the topology t
- **rest_server_v5.py** : to define the port and address for Pathman

To launch Pathman, use the command **python3 rest_server_v5.py** in the folder. You can now access via your browser to **localhost:8020/pathman/client/pathman/index.html#**

Here is the final topology of our network with every routers and links (Figure 6). On the top right corner, you can click on the double arrow to open the research of shortest path. You just have to enter a source router and a destination and press “Go”. A list of every path with every cost appears.

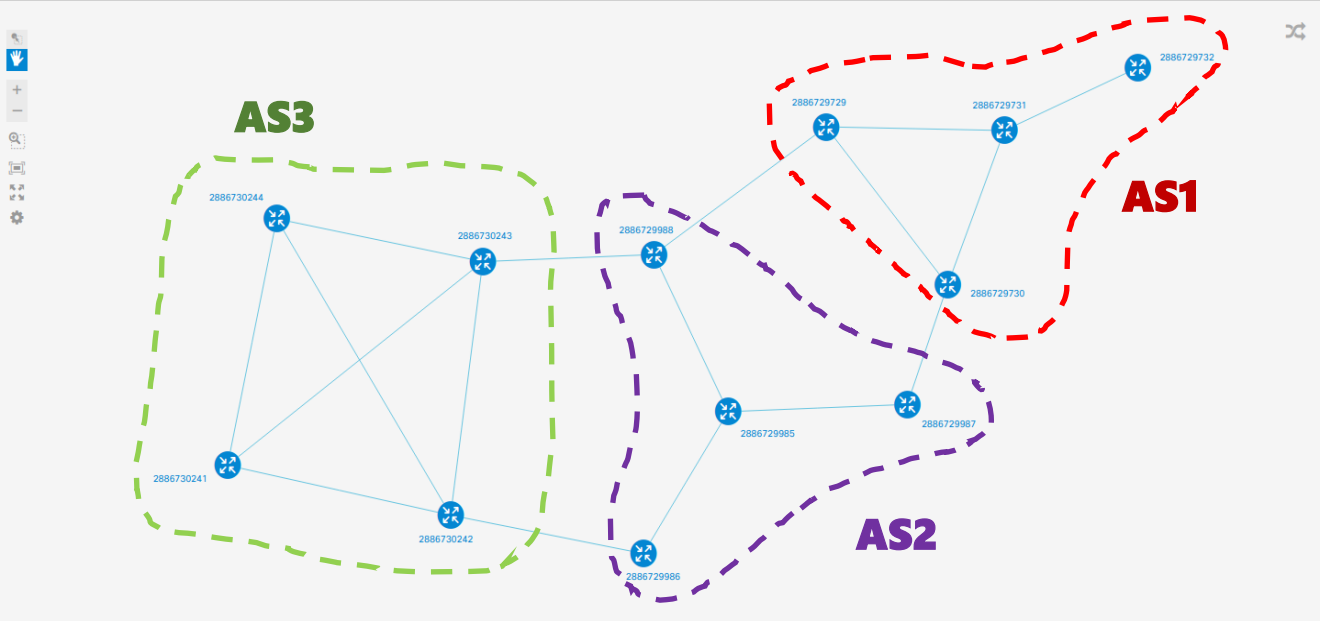


Figure 6 Topology displayed with Pathman

You can also go to **localhost:8020/pathman/client/BGP/index.html#**, which uses the BGP protocol to obtain information. In this interface you cannot find paths but there is more information about the different links of the network, for each router.

IV. Path determination and optimization

The last step of our project was divided in two parts : to find the shortest paths through the network and to optimize those paths.

A. Shortest paths determination

First, we wanted to have a list of several shortest paths for each demand (source – destination with a size). For this, we used the k-shortest path algorithm, a variant of the Bellman-Ford algorithm, but storing all the paths at each pass and not only the best one.

We have implemented this algorithm using Java code and the JgraphT library. After having retrieved the topology, this library allowed us to build the graph object and to use algorithms.

Our Java Code is situated in **/home/Documents/REST_MA/**

In the Main code we built the graph and we displayed it with another library (Graphstream) before running the algorithm (k-shortest path).

B. Optimization

For each optimization, we had to give to the functions the list of k-shortest paths that we have found at the previous step.

1. Minimization of the maximum utilisation

The first optimization we could do was the minimization of the maximum utilisation, because we do not want to use only one link and to saturate all its bandwidth. For the optimization, we used a Java Library, CPLEX.

2. Minimization of the cost

Finally, another possible optimization is about the cost of each link, to choose the path with the minimum cost possible.

C. LSP updating

After optimizing the paths for each demand, we had to notify our network of those different paths and demands. For this we had to create a new HTTP request.

CONCLUSION

To conclude, our project was aimed at creating a Java program that can help us create optimized routes using linear algebra and the K-Shortest path algorithm. That program will also be able to send back an LSP message to influence the routing information on the network.

Even if we were not experts on the domain of networking and routing, we managed to accomplish our objectives and learned a lot about those two subjects during this project.

We hope that our explanations were clear and understandable, and that the next person to continue this project can use what we created.

ANNEXES

Annex A : Cisco router configuration

```
!! IOS XR Configuration 6.0.1
!! Last configuration change at Fri Aug 17 16:01:22 2018 by cisco
!
hostname 3xrv3
ipv4 unnumbered mpls traffic-eng Loopback0
interface Loopback0
 ipv4 address 172.16.2.3 255.255.255.255
!
interface MgmtEth0/0/CPU0/0
 ipv4 address 192.168.100.123 255.255.255.0
!
interface GigabitEthernet0/0/0/0
 ipv4 address 10.10.2.6 255.255.255.252
!
interface GigabitEthernet0/0/0/1
 ipv4 address 10.10.2.22 255.255.255.252
!
interface GigabitEthernet0/0/0/2
 ipv4 address 10.10.2.13 255.255.255.252
!
interface GigabitEthernet0/0/0/3
 ipv4 address 192.168.148.4 255.255.255.0
!
interface GigabitEthernet0/0/0/4
 ipv4 address 192.168.222.1 255.255.255.252
!
route-policy pass-all
  pass
end-policy
!
router ospf 300
 area 0
  mpls ldp auto-config
  mpls traffic-eng
  interface GigabitEthernet0/0/0/4
   cost 60
   network point-to-point
  !
 !
 area 300
  mpls ldp auto-config
  mpls traffic-eng
  interface Loopback0
   cost 1
   passive enable
  !
  interface GigabitEthernet0/0/0/0
```

```
   cost 10
   network point-to-point
  !
  interface GigabitEthernet0/0/0/1
   cost 10
   network point-to-point
  !
  interface GigabitEthernet0/0/0/2
   cost 10
   network point-to-point
  !
 !
 mpls traffic-eng router-id Loopback0
!
rsvp
 interface GigabitEthernet0/0/0/0
  bandwidth 100000
 interface GigabitEthernet0/0/0/1
  bandwidth 100000
 interface GigabitEthernet0/0/0/2
  bandwidth 100000
 interface GigabitEthernet0/0/0/4
  bandwidth 50000
!
mpls traffic-eng
 interface GigabitEthernet0/0/0/0
 interface GigabitEthernet0/0/0/1
 interface GigabitEthernet0/0/0/2
 interface GigabitEthernet0/0/0/4
 pce
  peer source ipv4 192.168.148.4
  peer ipv4 192.168.148.1
  !
  stateful-client
   instantiation
   timers redelegation-timeout 0
   timers state-timeout 0
   delegation
  !
 !
 auto-tunnel pcc
  tunnel-id min 1 max 99
 !
!
mpls ldp
 router-id 172.16.2.3
 address-family ipv4
 !
!
end
```