# Desempenho e Dimensionamento de Redes

## Network Traffic Engineering

---

Python references: NetworkX - https://simpy.readthedocs.org/en/latest/

NetworkX Tutorial - https://networkx.readthedocs.org/en/stable/tutorial/

# Network and Traffic Definition

1. **Based** on the code provided in the python script **NetGen.py**, generate two networks: (i) a small 4-nodes network for testing/debugging purposes, and (ii) a large 18-nodes network for traffic engineering deployment. The generation processed includes the definition of the nodes (name and geographic location), the inter-node links, and the traffic matrix that defines the traffic flows between all cities/nodes. The geographic location (*pos*) is dynamically obtained from the node name (city name) using the Google Maps API (see the provided getGeo.py package). The traffic matrix (*tm*) is randomly generated. The network parameters are saved in a data file (default network.dat).

Run:

```
python NetGen.py -f smallnet.dat
```

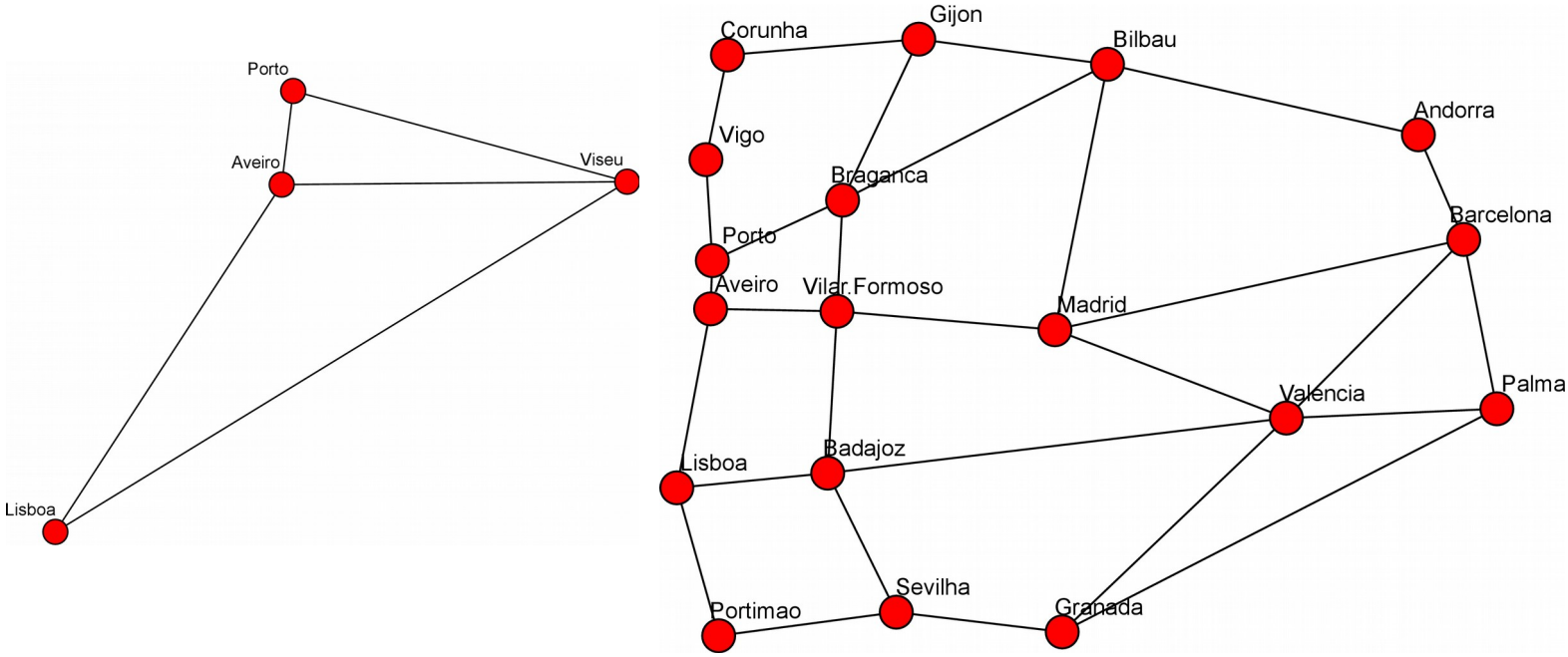
to obtain the small network (saved on file **smallnet.dat**).

Comment lines 20 and 21 (*nodes* and *links* for the small network), and run:

```
python NetGen.py
```

to obtain the large network (saved on file **network.dat**).

For both networks analyze the saved data structures: *nodes, links, pos, tm.*




# Traffic Engineering

## General Assumptions

- Consider all connections have a bandwidth capacity of 1 Gbps and the propagation speed is the light speed ($3\times10^8$ meters per second). Nodes have a very high routing speed (routing delay can be considered zero).

- Consider that networks support MPLS (Multi-Protocol Label Switching) and that the each traffic flow must be routed through a single LSP (Label Switched Path).

- Each connection is bi-directional and, therefore, has two load values, one on each of its directions.

- Assume that in all traffic flows, the packet arrivals are Poisson processes and the packet sizes are exponentially distributed with an average size of 1000 Bytes.

2. **Based** on the code provided in the python script **NetTE1.py,** and assuming that the routing of each flow/LSP is done using the shortest path given by the sum of the lengths of the connections, use a **Greedy Algorithm** to find a routing solution (1) to all flows/LSPs on the large network.

For the small network run: `python NetTE1.py -f smallnet.dat`

And for the large network run: `python NetTE1.py -f network.dat`

3. For the achieved routing solution, determine the resulting (i) average one-way delay of each flow and (ii) the load in both directions for all links.

Note: the one-way delay should be inferred based on the Kleinrock approximation.

4. For the achieved routing solution, determine (i) the average and the maximum link load, (ii) the average and the maximum one-way delay, and (iii) the flow with the worst QoS (in terms of one-way delay).

5. **Create** a python script named **NetTE2.py**, that uses a **Greedy Algorithm** to find a routing solution (2) to all flows/LSPs on the large network that, starting from an empty network, for each flow, chooses the routing path that minimizes the sum of the link loads that resulted from the previous selected routing paths.

6. For the achieved routing solution, determine the resulting (i) average one-way delay of each flow and (ii) the load in both directions for all links, (iii) the average and the maximum link load, (iv) the average and the maximum one-way delay, and (v) the flow with the worst QoS (in terms of one-way delay).

7. Analyzing the results of both solutions, identify which solution is the best and explain why.

8. **Create** a python script named **NetTE3.py**, that uses a **Greedy Algorithm** to find a routing solution (3) to all flows/LSPs on the large network that, starting from an empty network, for each flow, chooses the routing path that minimizes the average one-way that resulted from the previous selected routing paths (use the M/M/1 approximation for the delay of each connection).

9. For the achieved routing solution, determine the resulting (i) average one-way delay of each flow and (ii) the load in both directions for all links, (iii) the average and the maximum link load, (iv) the average and the maximum one-way delay, and (v) the flow with the worst QoS (in terms of one-way delay).

10. Analyzing the results of all three solutions, discuss which solution is better.

11. Solutions (2) and (3) depend greatly on the order of the LSP allocation. **Create** a python script named **NetTE4.py** based on NetTE3.py, where in each run the order of the flows/LSPs allocation is randomly chosen.

Use the following instruction to randomize the initial order of the LSPs (order of the pairs of nodes):

```
random.shuffle(allpairs)
```

Run multiple times and compare the obtain results/solutions.

Note: this algorithm is called **Greedy Randomized Algorithm.**

12. Upgrade the script **NetTE4.py** to implement multiple runs of the **Greedy Randomized Algorithm** and search for the best solution (after a limited number of runs) for the flows/LSPs allocation on the network, minimizing the lowest network average one-way delay.

13. **Create** a python script named **NetTE5.py**, to implement a **Local Search Algorithm** to search for the best solution for the flows/LSPs allocation on the network, minimizing the lowest network average one-way delay.

Start by choosing an initial solution, e.g., solution (1). Create the set of neighbor solutions, where each neighbor solution can be found by removing an individual flow from the initial solution and reallocation the flow using the criteria used for solutions (2) and/or (3). Finally, choose the solution with the lowest network average one-way delay.

14. **Create** a python script named **NetTE6.py**, to implement a **Multi-Start Local Search Algorithm** to search for the best solution for the flows/LSPs allocation on the network, minimizing the lowest network average one-way delay.

Run in a cycle until a stop criteria is met:

1. Create a random base solution using a **Greedy Randomized Algorithm.**

2. Run a **Local Search Algorithm** starting form the base solution.
3. If the cycle solution is better than the best overall solution found so far, the cycle solution is stored as the overall best solution.
4. Go to step 1.