Universidad Politécnica de Cartagena

TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2015-2016

Lab work #3. Introduction to Net2Plan algorithms development (II)

(1 session)

Author:

Pablo Pavón Mariño

1 Objectives

The goals of this lab work are:

1. Introduce the development of more complex Net2Plan offline design algorithms, getting more familiar with the Javadoc documentation.

2 Duration

This lab work is designed for one session of two hours.

3 Evaluation

This lab work has been designed to guide the students in their learning of Net2Plan. The annotations the students make in this document are for their use when studying the course, and do not have to be delivered to the teacher for evaluation.

4 Documentation

The resources needed for this lab work are:

- Net2Plan tool and their documentation (see `http://www.net2plan.com/`).
- Instructions in this wording.

5 Previous work before coming to the lab

- Make the *quizs* left unfinished from the previous lab session.

6 Shortest-path routing and capacity allocation

Important: Before starting this lab work, the student should open in separate browsers the documentation that will need along the session:

- Net2Plan users guide.
- Net2Plan Javadoc documentation.
- Standard Java Javadoc documentation.

Open Eclipse and create a new project to host the new algorithm to develop. Copy the template file `AlgorithmTemplate.java` and rename it as `LabSession3_cfa.java`. The student should modify this file, to implement an algorithm with the following characteristics:

- The algorithm has a single input parameter called *cg*, with default value equal to $cg = 0.6$. It is the utilization that will be enforced in all the links.
- The algorithm receives an input design which is supposed to have nodes, links and traffic demands (unicast). With this design the algorithm should:
  - Set the routing type as *source-routing*.

    Use the `setRoutingType` method of the *NetPlan* object (see the Javadoc!!!).
  - Remove any input route or protection segment.

    Use the `removeAllRoutes` and `removeAllProtectionSegments` methods of the *NetPlan* object (see the Javadoc!!!).
  - For each demand, the traffic should be carried using a single *Route*, where the link capacity occupied by the route is the same as the traffic carried, and the route path is the **shortest path** in number of hops (number of traversed links) between the demand end nodes.

    For computing the sequence of *Link* objects that corresponds to the shortest path between two nodes, use the *getShortestPath* method in the *GraphUtils* class of package *com.net2plan.libraries* (see the Javadoc!!!).
  - After previous point in completed, the network links are carrying the routed traffic. Now, for each link, set its capacity so that its utilization becomes equal to the input parameter *cg*. This means that the capacity u_e of a link e should be made equal to $u_e = y_e/cg$, where y_e is the link capacity currently occupied by the carried traffic.

    To see the occupied capacity by the traffic in a particular link, use the method *getOccupiedCapacityIncludingProtectionSegments* in the *Link* object (see the Javadoc!!!).

    To set the capacity of a link, use the method *setCapacity* of the *Link* object (see the Javadoc!!!).
- The output message of the algorithm should include the total amount of capacity installed in the links (that is, the sum of the link capacities).

6.1 Checking the algorithm

The student can check its implementation by loading the network `NSFNet_N14_E42_complete.n2p`, and then running the algorithm in it with the default input parameter *cg=0.6*:

- The resulting total link capacity should be 13866.090 (see the *Layer* statistics for this).
- All the links should have an utilization equal to 0.6.
- The design should have no protection segment.

7 Now on your own

The target of these exercises is letting the student get familiar with the Javadoc documentation, and the most typically used methods in the classes *NetPlan*, *Node*, *Link*, *Demand*, etc.

Quiz 1. Create an algorithm that removes all the existing demands in the network, and then creates one demand between each node pair, with a constant offered traffic *traf*, an input parameter of the algorithm with default value 1.

Quiz 2. Create an algorithm that removes all the existing routes in the network. Then, it goes through all the traffic demands of the design, in increasing order of index (0,1,2,...). For each demand, the algorithm should use the method *getCapacitatedShortestPath* in *GraphUtils* class, to compute the shortest path between the demand end nodes, **that has a sufficient unused capacity in the links to carry the traffic of the new demand**. If no path is returned, the demand is blocked. If a path is found, we route all the demand traffic through it. As usual, the link occupied capacity is equal to the link carried traffic. Hints:

- For iterating through the demands you can use any of the two following options:

```
1  for (int indexD = 0; indexD < netPlan.getNumberOfDemands() ;
       indexD ++)
2  {
3    Demand d = netPlan.getDemand (indexD);
4    ...
5  }
```

or

```
1  for (Demand d : netPlan.getDemands())
```

Since the latter iterates the demands also in increasing order of indexes, starting from 0.

- To see the occupied capacity by the traffic in a particular link, use the method *getOccupiedCapacityIncludingProtectionSegments* in the *Link* object (see the Javadoc!!!).

8 Work at home after the lab work

The student is encouraged to complete all the *Quizs* that he/she could not finish during the lab session.