Universidad Politécnica de Cartagena

TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2018-2019

Práctica 3. Introducción al desarrollo de algoritmos de Net2Plan (II)

(1 sesión)

Autor:

Pablo Pavón Mariño
Juan Pedro Muñoz Gea
María Victoria Bueno Delgado
Ángel Antonio Pintado Sedano
Juan Carlos Sánchez Aarnoutse

1. Objetivos

Los objetivos de esta práctica son:

1. Introducir al alumno en el desarrollo de diseños más complejos de algoritmos de Net2Plan, familiarizándose cada vez más con la documentación del Javadoc.

2. Duración

Esta práctica está diseñada para una sesión de dos horas.

3. Evaluación

Esta práctica ha sido diseñada para guiar al estudiante en su aprendizaje en Net2Plan. Las anotaciones que los estudiantes hagan en este documento son para su uso cuando repasen la asignatura y no es necesario que se entregue al profesor para su evaluación.

4. Documentación

Los recursos necesarios para este trabajo de laboratorio son:

- La herramienta Net2Plan y su documentación (ver `http://www.net2plan.com/`).
- Las instrucciones de esta práctica.

5. Trabajo previo antes de venir al laboratorio

- Haber realizado los *quizes* no terminados en la práctica anterior.

6. Enrutamiento por el camino más corto y asignación de capacidad

Importante: Antes de comenzar esta práctica, el alumno debe abrir en navegadores separados la documentación que necesitará a lo largo de la sesión:

- Guía de usuario Net2Plan.
- Documentación del Javadoc de Net2Plan.
- Documentación estandar Java del Javadoc.

Abra Eclipse y cree un nuevo proyecto para albergar el nuevo algoritmo a desarrollar. Copie la plantilla `AlgorithmTemplate.java` y cambie el nombre a `LabSession3_cfa.java`. El alumno debería modificar este fichero para implementar un algoritmo con las siguientes características:

- El algoritmo tiene un único parámetro de entrada llamado *cg*, con un valor por defecto de $cg = 0{,}6$. Este valor será el factor de utilización que se fijará en todos los enlaces.
- El algoritmo recibe un diseño de entrada que se supone que tiene nodos, enlaces y demandas de tráfico (unicast). Con este diseño el algoritmo debería:
  - Fijar el tipo de enrutamiento como *source-routing*.

    Utilizar el método `setRoutingType` del objeto *NetPlan* (ver el Javadoc!!!).
  - Eliminar cualquier ruta de entrada.

    Utilizar el método `removeAllRoutes` del objeto *NetPlan* (ver el Javadoc!!!).
  - Para cada demanda el tráfico debería cursarse utilizando una única *Ruta*, donde la capacidad del enlace ocupada por la ruta es la misma que el tráfico cursado, y el camino utilizado es el **shortest path** en número de saltos (número de enlaces atravesados) entre los nodos extremos de la demanda.

    Para calcular la secuencia de objetos *Link* que corresponde al camino más corto entre dos nodos, utilice el método *getShortestPath* en la clase *GraphUtils* del paquete *com.net2plan.libraries* (ver el Javadoc!!!).
  - Después de completar el punto anterior, los enlaces de la red estarán cursando el tráfico a enrutar. Ahora, para cada enlace, establezca su capacidad de modo que su utilización sea igual al parámetro de entrada *cg*. Esto significa que la capacidad u_e de un enlace e debería ser igual a $u_e = y_e/cg$, donde y_e es la capacidad del enlace actualmente ocupada por el tráfico cursado.

    Para ver la capacidad ocupada por el tráfico en un enlace en particular, utilice el método *getOccupiedCapacity* en el objeto *Link* (ver el Javadoc!!!).

    Para fijar la capacidad de un enlace utilice el métdodo *setCapacity* del objeto *Link* (ver el Javadoc!!!).
- El mensaje de salida del algoritmo debería incluir la capacidad total instalada en los enlaces (es decir, la suma de las capacidades de enlace).

6.1. Comprobando el algoritmo

Los alumnos pueden comprobar su implementación cargando la red `NSFNet_N14_E42_complete.n2p`, y ejecutando el algoritmo sobre ésta con el valor de entrada por defecto *cg=0.6*:

- La capacidad resultante total debe ser de 13866.090 (ver el *Layer* estadísticas para ello).
- Todos los enlaces deben tener una utilización igual a 0.6.

7. Ahora por su cuenta

El objetivo de este ejercicio es hacer que los estudiantes se familiaricen con la documencación Javadoc, así como los métodos más empleados habitualmente en las clases *NetPlan*, *Node*, *Link*, *Demand*, etc.

Quiz 1. Cree un algoritmo que elimine todas las demandas existentes en la red y que después cree una demanda entre cada par de nodos con un tráfico ofrecido constante *traf*. Esta constante debe tener un valor por defecto de 1.

Quiz 2. Cree un algoritmo que elimine todas las rutas existentes en la red. A continuación, el algoritmo debe recorrer todas las demandas de tráfico del diseño en order creciente del índice (0,1,2,...). Para cada demanda, el algoritmo debe emplear el método *getCapacitatedShortestPath* en la clase *GraphUtils* para calcular el camino más corto entre los nodos finales demandantes, **que tenga suficiente capacidad sin emplear en los enlaces como para cursar el tráfico de la nueva demanda**.Si no devuelve un camino, la demanda se debe bloquear. Si encuentra un camino, se encamina todo tráfico de demanda a través de él. Como es habitual, la capacidad ocupada del enlace es igual al tráfico cursado por el enlace. Pistas para el desarrollo:

- Para itinerar entre las demandas, puede emplear cualquiera de las siguientes opciones:

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

o

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

Nótese que la última opción también itera en orden creciente de índices, comenzando por 0.

- Para ver la capacidad ocupada por el tráfico en un enlace concreto emplee el método *getOccupiedCapacityIncludingProtectionSegments* contenido en el objeto *Link* (vea la documentación Javadoc!!!).

8. Trabajo en casa después de la sesión de laboratorio

El estudiante debería completar todos los *Quizs* que no haya podido finalizar durante la sesión en el laboratorio.