TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2015-2016

Lab work #8. Congestion control

(1 session)

Author:

Pablo Pavón Mariño

1 Objectives

The goals of this lab work are:

1. Create Net2Plan algorithms that solve formulations of the congestion control problem using the *Java Optimization Modeler* (JOM) library.
2. Gain experience with the different forms of writing optimization problems in JOM, making benefit of its vectorial representation capabilities.

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:

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

5 Previous work before coming to the lab

- Read Section 3.8 and 6.2 of [1], and the lecture notes regarding congestion control.
- Refresh your reading in the JOM documentation in `http://www.net2plan.com/jom`, in particular, how vector of variables and constraints are handled.

6 Estimating the performance of TCP Reno sources

Let $\mathcal{G}(\mathcal{N},\mathcal{E})$ be a network, where the set of nodes $\mathcal{N}$ and the set of links $\mathcal{E}$ are given. Link capacities $(u_e, e \in \mathcal{E})$ are known. The offered traffic is composed of a set of unicast demands $\mathcal{D}$. For each demand $d \in \mathcal{D}$, the sequence of traversed links p_d is known, but the amount of offered traffic h_d is unknown.

We assume that each demand represents one unidirectional flow of a TCP elephant connection (that is always willing to transmit traffic), using a TCP-Reno version. As we have seen in the lectures, the NUM model (*Network Utility Maxmimization*) can be used to estimate the average (in macroscopic equilibrium) injected traffic of each connection. In particular, the TCP connection average rates are given by the optimum solution of the following formulation:

- Input parameters (known constants):
  - $\mathcal{N}$: Set of network nodes.
  - $\mathcal{E}$: Set of network links.
  - $u_e, e \in \mathcal{E}$: Capacity of link e.
  - $\mathcal{D}$: Set of offered unicast demands (TCP connections, considered unidirectional).
  - $p_d, d \in \mathcal{D}$: Sequence of traversed links of the TCP connection d. We denote as $\mathcal{P}_e$ to the set of TCP connections that traverse a given link e.
  - $RTT_d, d \in \mathcal{D}$: Round-trip-time of the packets in connection d. We assume here that a connection RTT considers only the propagation time of the traversed links (multiplied by two, to account for the return path).
- Decision variables:
  - $h_d, d \in \mathcal{D}$: Average traffic injected by TCP connection d.
- Formulation:

$$\max \quad -\sum_{d} \frac{3}{2RTT_d^2 h_d}, \quad \text{subject to:} \tag{1a}$$

$$\sum_{d \in \mathcal{P}_e} h_d \leq u_e, \quad \forall e \in \mathcal{E} \tag{1b}$$

$$h_d \geq 0, \quad \forall d \in \mathcal{D} \tag{1c}$$

The objective function (1a) represents the NUM model, that tries to maximize the sum of the utilities of each TCP connection. The utility function $U_d(h_d)$ of a TCP Reno connection d, according to the model seen in theory, is given by:

$$U_d(h_d) = -\frac{3}{2RTT_d^2 h_d}$$

Constraints (1b) are the standard link capacity constraints, and mean that for each link, the traffic carried in the link is less or equal than its capacity (and thus, no link is oversubscribed). Finally, (1c) forbid injecting a negative amount of traffic.

7 Net2Plan algorithm

The student should develop a Net2Plan algorithm solving problem (1) following the next steps:

1. Copy the `AlgorithmTemplate.java` file in Aula Virtual and rename it as `TCPReno.java`.
2. Set the routing type of the input `NetPlan` object to source routing.
3. Eliminate any carried traffic by removing all the routes in the network.

4. For each unicast demand, create a route for it carrying zero traffic, traversing the shortest path between the demand end nodes measured in number of hops (e.g. using `GraphUtils` method `getShortestPath`).

5. Create an object of the type `OptimizationProblem` (e.g. of name `op`).

6. Add the problem decision variables, with name `h_d`: one variable per *route*. The i-th coordinate corresponds to the `Route` object of index i (which is associated to a specific demand). The minimum value of the variables is set to zero, the maximum to `Double.MAX_VALUE`.

7. Set the problem objective function. For this, you can use the method of `NetPlan` class:

   ```
   netPlan.getVectorRoutePropagationDelayInMiliseconds
   ```

   to obtain a `DoubleMatrix1D` vector with the propagation delay (measured in ms) of each route, which can be converted to a standard `double[]` array using the method `toArray`. Recall that the round-trip time of a connection is twice its one-way propagation delay[1]. The student is suggested to create a vector with the coefficients: $c_d = -\frac{3}{2RTT_d^2}$, add it as an input parameter to the problem, and make the objective function to maximize becomes:

   ```
   sum (z_d ./ h_d).
   ```

   where `./` is the element-by-element division of two arrays which must have the same size.

8. Use a `for` loop with as many iterations as links, to add the link capacity constraints (1)b. For adding the constraint of a link `e`:

   - Set the JOM input parameters:
     - `P_e` with the indexes of the routes traversing link `e`. For this, use the method *getTraversingRoutes* of the node object to get the output links, and the method *NetPlan.getIndexes* to convert the collection of links to their indexes.
     - `u_e` with the capacity of the link.
   - Set the constraint using the function `sum`, over `h_d`, but restricting the sum to the elements in `P_e`.

9. Call the solver to find a numerical solution. Since the problem is not linear, the solver to use is `ipopt`.

10. Retrieve the primal solution obtained.

11. Save it in the `netPlan` object. For this, use a `for` loop iterating along the routes of the design. For each route: (i) set the route carried traffic and occupied link capacity according to the optimum solution, (ii) set the offered traffic of the associated demand to be equal to the carried traffic.

7.1 Check the algorithm

Load the network `example7nodesWithTraffic.n2p`. The algorithm should produce a solution with a total of 600.36 units of total offered traffic, and all the links should have a utilization of 100%.

[1]Note that multiplying the objective function by any positive constant does not chage the optimum. Then, constant factors can be removed from the objective function. Also, the result would not change is we measure the RTT in seconds or any other unit.

8 Problem variations

Quiz 1. Modify the problem in (1) so now the cost objective function becomes the well known α-fair funtions:

$$\sum_d \frac{h_d^{1-\alpha}}{1-\alpha}$$

for $\alpha \geq 0, \alpha \neq 1$, and

$$\sum_d \log h_d$$

when $\alpha = 1$. Implement a Net2Plan algorithm that has an input parameter of name `alpha` (defaults to 1), and solves the NUM formulation with α factor equal to the input parameter `alpha`. To check the solution, load the network `example7nodesWithTraffic.n2p`. The throughput for $\alpha = 2$ is 514.64, and for $\alpha = 1$ is 549.26.

Quiz 2. Use the algorithm you have just developed to fill in the following table, which shows how the network throughput varies when different α fairness values are used by the congestion control.

The table refers to NSFNET network in the file `NSFNet_N14_E42.n2p`, for link capacities $u_e = 500$, and one demand per each node pair[2]. As a check, we include the results for $\alpha = 2$.

NSFNET topology

α	**Jain fairness factor**	**Throughput $\sum_d h_d$**
0		
0.5		
1		
2	0.48	12311
5		

Column *Jain fairness factor* should display the well-known Jain factor J (see e.g. "Jain's fairness measure" in `http://en.wikipedia.org/wiki/Fairness_measure`). Factor J is a measure of fairness in the distribution of resources. In our case, J is given by:

$$J = \frac{\left(\sum_d h_d\right)^2}{|\mathcal{D}| \times \sum_d h_d^2}$$

where $|\mathcal{D}|$ is the number of demands. Jain factor will be maximum ($J = 1$) when all the demands have exactle the same rate ("maximum fairness"), and will take the mininum possible value ($J = 1/|\mathcal{D}|$) when one demands receives traffic and the rest receive nothing ("minimum fairness").

The student should include in the algorithm the code that computes and prints in `System.out` the J index for the obtained solution.

- Are there demands with 0 rate assigned in the case $\alpha = 0$? could a network like Internet be useful if the congestion control was designed to maximize the throughput?
- What is the trend in throughput and in fairness when we increase α?

[2] *Note*: For high α values (e.g. $\alpha \geq 4$), solutions can differ in different executions because of numerical instabilities in the solver.

9 Matricial form of problem constraints (optional)

9.1 Link capacity constraints

The link capacity constraints in (1b) take the form:

$$\sum_{d:e\in \surd^{d}} h_d \leq u_e, \forall e \in \mathcal{E}$$

All the $|\mathcal{E}|$ constraints can be represented by a single vectorial inequality as follows:

$$A_{ep} \times h'_d \leq u'_e \tag{2}$$

where:

- A_{ep} is link-to-route assignment matrix. This is a $|\mathcal{E}| \times |\mathcal{P}|$ matrix with one row per link, and one column per route (in this case, one per demand). Coordinate (e, p) is the number of times that path p traverses link e.
  - The link-to-route assignment matrix can be obtained as an sparse matrix in Net2Plan using the method of `NetPlan` class:

    ```
    getMatrixLink2RouteAssignment
    ```

- h_d is a $1 \times |\mathcal{D}|$ row vector, with the decision variables (the traffic carried by route associated to demand d).
- u_e is a row vector $1 \times |\mathcal{E}|$ with the capacity of each link.
  - The link capacity vector can be obtained in Net2Plan using the method of `NetPlan` class:

    ```
    getVectorLinkCapacity
    ```

Quiz 3. Rewrite the link capacity constraints using their matricial form. Recall that JOM operator `*` implements the standard matrix multiplication.

10 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.

Bibliography

[1] *P. Pavón Mariño, "Optimization of computer networks. Modeling and algorithms. A hands-on approach", Wiley 2016.*