DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

MESTRADO INTEGRADO EM ENG. DE COMPUTADORES E TELEMÁTICA

ANO 2018/2019

DESEMPENHO E DIMENSIONAMENTO DE REDES

ASSIGNMENT GUIDE NO. 5

TRAFFIC ENGINEERING OF TELECOMMUNICATION NETWORKS

Assignment Guide No. 5

1. Preamble

The aim of this assignment is to address the traffic engineering of an ISP (Internet Service Provider) core network based on MPLS (Multi-Protocol Label Switching).

For a given network and a given set of estimated traffic flows to be supported, the traffic engineering of this assignment is the task of selecting a routing path for the LSP (Label Switched Path) of each traffic flow. The aim of the task is to obtain different solutions providing different trade-offs between the resilience to unpredictable traffic growth and the energy consumption of the network.

2. General Description

Consider the MPLS (Multi-Protocol Label Switching) network of an ISP (Internet Service Provider) with the topology of Figure 1 where:

- the gray nodes are transit nodes (they just provide connectivity between the other nodes),
- the links highlighted with thick lines have a capacity of 10 Gbps while the other links have a capacity of 1 Gbps;
- the values close to each link are the energy consumption of each link when it is in the active mode (the links without the value have an energy consumption of 1).

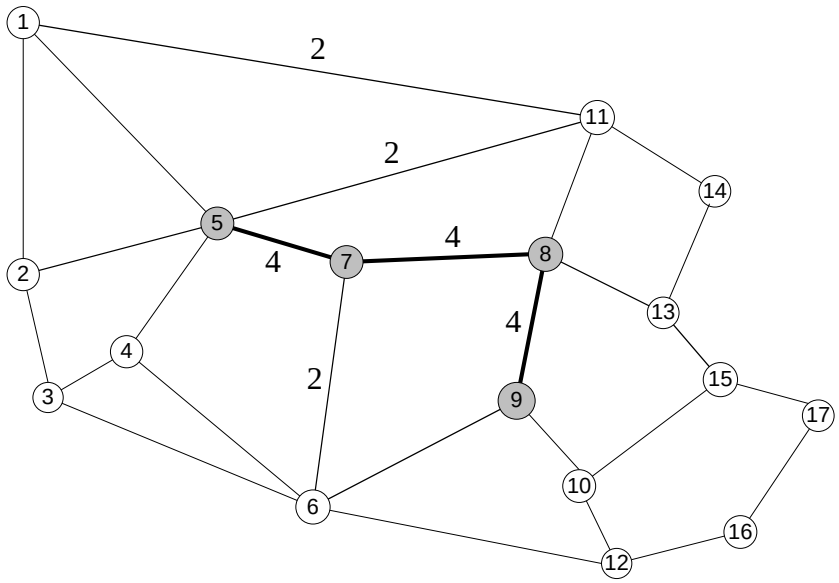

Figure 1: Topology of an ISP Network

In the provided MATLAB file named `Matrices.m`, the *R*, *C* and *L* matrices define the network topology of Figure 1 where elements $R(i,j)$, $C(i,j)$ and $L(i,j)$ provide the capacity (in Gbps), the energy consumption and the length, respectively, of the connection from node *i* to node *j* (when there is no link between *i* and *j*, $R(i,j) = 0$, $C(i,j) = 0$ and $L(i,j) =$ Infinite).

The network must support an estimated traffic matrix. In the provided `Matrices.m`, the *T* matrix defines the traffic matrix where element $T(i,j)$ defines the average traffic bandwidth (in Mbps) sent from node *i* to node *j*.

Consider the following operational properties:

- The operator requires each traffic flow to be routed through a single LSP (Label Switched Path). The LSPs between the same nodes are not required to be symmetrical (*i.e.*, the links of the LSP from node *i* to node *j* do not need to be the same links of the LSP from node *j* to node *i*).
- Each link can be put in a sleeping mode if it is not supporting flows in neither of its directions and sleeping mode links have a negligible energy consumption.

Develop appropriate MATLAB codes to solve the 2 following assignments.

3. Minimizing Worst Link Load

The aim is to compute a routing path for each flow such that the worst link load is minimized. Note that each link has two link loads, one on each direction: the link load on a direction of a link is given by the total bandwidth of all flows routed through the link direction divided by the link capacity.

Annex I provides a MATLAB code to compute the candidate routing paths for each flow. Annex II shows the provided MATLAB function `maxLoad` which computes the worst link load of a routing solution.

Develop a local search algorithm to find the best solution using the best neighbor move variant. As its initial solution, consider the assignment of the shortest routing path to each flow. Annex III shows a MATLAB script provided in file `main.m` to compute this initial solution. Use this script as starting point to the development of your local search algorithm.

When computing the candidate routing paths, consider the *k*-shortest path algorithm for $k = 5, 10, 15, 20, 25$ and 30. Run the local search algorithm for each value of *k* and register: (i) the obtained worst link load and (ii) the total local search running time. Draw conclusions on the obtained results. What is the value of *k* that gives the best performance?

4. Minimizing Energy Consumption for a Given Worst Link Load

For a given worst link load value *w*, the aim is to compute a routing path for each flow such that the worst link load is not higher than *w* and the energy consumption is minimized. Consider the worst link load values $w = 0.7, 0.75, 0.80, 0.85$.

Develop a greedy algorithm to select each link to be put in sleeping mode. At each step, run the (previously developed) local search algorithm using the best value of *k*. Consider the greedy algorithm with 2 selection criteria: (i) the link providing the lowest worst link load and (ii) the link providing the best ratio of the worst link load divided by the energy consumption of the link.

Among the solutions provided by both selection criteria, identify the best solutions for each value of *w*.

ANNEX I – COMPUTING THE CANDIDATE ROUTING PATHS

Consider the following MATLAB code to compute the average flow bandwidth and the candidate routing paths of each flow:

```
nNodes= size(L,1);
% nPaths is the number k of candidate paths to run the k-shortest path
% algorithm:
nPaths= 20;
f= 0;
for i=1:nNodes
    for j= 1:nNodes
        if T(i,j)>0
            f= f+1;
            flowDemand(f) = T(i,j);
            % kShortestPath implements the k-shortest path algorithm:
            [shortestPaths{f}, tc] = kShortestPath(L,i,j,nPaths);
            if isempty(tc)
                fprintf('Error: no connectivity\n');
            end
        end
    end
end
```

At the end, `flowDemand` is an array with the average flow bandwidth of each flow and `shortestPaths{f}` is a cell array with all routing paths of flow `f`.

The number of routing paths of flow `f` is given by `length(shortestPaths{f})`. The k^{th} routing path is provided by `shortestPaths{f}{k}` as a row vector of the sequence of nodes of the routing path. The first node is `shortestPaths{f}{k}(1)`, the second node is `shortestPaths{f}{k}(2)`, …, the last node is `shortestPaths{f}{k}(end)`.

Note that the function `kShortestPath(L,i,j,nPaths)` might return less than `nPaths` routing paths if the total number of possible paths is less than `nPaths`.

ANNEX II – COMPUTING THE WORST LINK LOAD

Consider the following MATLAB function (provided in file `maxLoad.m`) to compute the worst link load of a routing solution:

```
function maximum= maxLoad(paths,shortestPaths,flowDemand,R)
%
% maximum= maxLoad(paths,shortestPaths,flowDemand,R)
% INPUTS:
% paths -         vector (size of no. of flows) with the path number
%                 selected for each flow
% shortestpaths - cell array with the k-shortest paths of each flow
% flowDemand -    vector (size of no. of flows) with the average flow
%                 bandwidth of each flow (in Mbps)
% R -             square matrix with capacity of each link (in Gbps)
%
% OUTPUT:
% maximum - the worst link load

nNodes=size(R,1);
nFlows= length(flowDemand);
aux= zeros(nNodes);
for f= 1:nFlows
    for k= 1:length(shortestPaths{f}{paths(f)})-1
        i= shortestPaths{f}{paths(f)}(k);
        j= shortestPaths{f}{paths(f)}(k+1);
        aux(i,j)= aux(i,j)+flowDemand(f);
    end
end
aux= aux./(1000*R);
aux(isnan(aux))= 0;
maximum= max(max(aux));
end
```

ANNEX III – COMPUTING THE SHORTEST PATH ROUTING SOLUTION

Consider the following MATLAB script (provided in file `main.m`) to compute the worst link load of the shortest path routing solution (i.e., each flow is routed through the shortest path route):

```
Matrices;      %load of matrices R, C, L and T in matrices.m
nNodes= size(L,1);
nPaths= 20;
f= 0;
for i=1:nNodes
    for j= 1:nNodes
        if T(i,j)>0
            f= f+1;
            flowDemand(f) = T(i,j);
            [shortestPaths{f}, tc] = kShortestPath(L, i, j, nPaths);
            if isempty(tc)
                fprintf('Error: no connectivity\n');
            end
        end
    end
end
nFlows= length(flowDemand);
%solution that considers the first candidate path for all flows:
solution= ones(1,nFlows);
worstlinkload= maxLoad(solution,shortestPaths,flowDemand,R)
```