Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Mestrado Integrado Eng. Computadores e Telemática

47064 - Desempenho e Dimensionamento de Redes

Relatório
Engenharia de trafego de uma rede com comutação de pacotes

Autores :
Guilherme Cardoso 45726
Rui Oliveira 68779

Docente :
Amaro Sousa

Prática :
P2

Ano letivo 2016/2017
Aveiro, 13 de Junho de 2017

Conteúdo

1 Introdução

Este relatório encontra-se estruturado da mesma forma que o guião, seguindo a mesma numeração dos exercícios propostos.

2 Implementação

2.1 Alínea a)

Todo o código para esta alínea foi fornecido pelo docente via elearning ou no próprio guia.

2.2 Alínea b)

2.2.1 Versão 1

```
Matrizes;
miu= R*1e9/(8*1000);            % capacidade em bits / pacotes de 1000bytes
    -> pacotes/sec
NumberLinks= sum(sum(R>0));     % numero total de ligacoes
lambda_s= T*1e6/(8*1000);       % packet arrival rate
gama= sum(sum(lambda_s));       % trafego total na rede (packets/sec)
d= L*1e3/2e8;                   % velocidade propagacao / vel da luz na
    fibra optica
pairs= [];

% descobrir nos ligados por um path bidirecional
for origin=1:16
    for destination=(origin+1):17
        if T(origin,destination)+T(destination,origin)>0
            pairs= [pairs; origin destination];
        end
    end
end
npairs= size(pairs,1);
lambda= zeros(17);
routes= zeros(npairs,17);

for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
    Loads= lambda./miu; % minimizar solucao; caga maxima menos significativa
    r= ShortestPathSym(Loads,origin,destination);
    routes(i,:)= r;
    j= 1;
```

```
    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_s(origin,
    destination);
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_s(destination,
    origin);
        j= j+1;
    end
end
Load= lambda./miu;
Load(isnan(Load))= 0;
MaximumLoad= max(max(Load))
AverageLoad= sum(sum(Load))/NumberLinks
AverageDelay= (lambda./(miu-lambda)+lambda.*d);
AverageDelay(isnan(AverageDelay))= 0;
AverageDelay= 2*sum(sum(AverageDelay))/gama
Delay_s= zeros(npairs,1);

for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
    r= routes(i,:);
    j= 1;
    while r(j)~= destination
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j),r(j+1)) - lambda(r(j),r(j+1))) +
    d(r(j),r(j+1));
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j+1),r(j)) - lambda(r(j+1),r(j))) +
    d(r(j+1),r(j));
        j= j+1;
    end
end

MaxAvDelay= max(Delay_s)
subplot(1,2,1)
printDelay_s= sortrows(Delay_s,-1);
plot(printDelay_s)
axis([1 npairs 0 1.1*MaxAvDelay])
title('Flow Delays')
subplot(1,2,2)
printLoad= sortrows(Load(:),-1);
printLoad= printLoad(1:NumberLinks);
plot(printLoad)
axis([1 NumberLinks 0 1])
title('Link Loads')
```

2.2.2 Versão 2

Detalhe da mudança feita da versão 1 para a versão 2.

```
(...)
```

```
2
3  for i=1:npairs
4
5      (...)
6
7      r= ShortestPathSym(Loads.^2,origin,destination);
8
9      (...)
10 end
11 (..)
```

2.3 Alínea c)

```
1  Matrizes;
2  miu= R*1e9/(8*1000);              % capacidade em bits / pacotes de 1000bytes
       -> pacotes/sec
3  NumberLinks= sum(sum(R>0));       % numero total de ligacoes
4  lambda_s= T*1e6/(8*1000);         % packet arrival rate
5  gama= sum(sum(lambda_s));         % trafego total na rede (packets/sec)
6  d= L*1e3/2e8;                     % velocidade propagacao / vel da luz na
       fibra optica
7  pairs= [];
8
9  % descobrir nos ligados por um path bidirecional
10 for origin=1:16
11     for destination=(origin+1):17
12         if T(origin,destination)+T(destination,origin)>0
13             pairs= [pairs; origin destination];
14         end
15     end
16 end
17 npairs= size(pairs,1);
18 lambda= zeros(17);
19 routes= zeros(npairs,17);
20
21 for i=1:npairs
22     origin= pairs(i,1);
23     destination= pairs(i,2);
24
25     aux = 1./(miu-lambda) + d; % M/M/1 Atraso m dio no sistema Modulo 2
       slides
26     r= ShortestPathSym(aux,origin,destination);
27
28     routes(i,:)= r;
29     j= 1;
30     while r(j)~= destination
31         lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_s(origin,
```

```
    destination);
32          lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_s(destination,
    origin);
33          j= j+1;
34      end
35  end
36  Load= lambda./miu;
37  Load(isnan(Load))= 0;
38  MaximumLoad= max(max(Load))
39  AverageLoad= sum(sum(Load))/NumberLinks
40  AverageDelay= (lambda./(miu-lambda)+lambda.*d);
41  AverageDelay(isnan(AverageDelay))= 0;
42  AverageDelay= 2*sum(sum(AverageDelay))/gama
43  Delay_s= zeros(npairs,1);
44
45  for i=1:npairs
46      origin= pairs(i,1);
47      destination= pairs(i,2);
48      r= routes(i,:);
49      j= 1;
50      while r(j)~= destination
51          Delay_s(i)= Delay_s(i)+ 1/(miu(r(j),r(j+1)) - lambda(r(j),r(j+1))) +
    d(r(j),r(j+1));
52          Delay_s(i)= Delay_s(i)+ 1/(miu(r(j+1),r(j)) - lambda(r(j+1),r(j))) +
    d(r(j+1),r(j));
53          j= j+1;
54      end
55  end
56
57  MaxAvDelay= max(Delay_s)
58  subplot(1,2,1)
59  printDelay_s= sortrows(Delay_s,-1);
60  plot(printDelay_s)
61  axis([1 npairs 0 1.1*MaxAvDelay])
62  title('Flow Delays')
63  subplot(1,2,2)
64  printLoad= sortrows(Load(:),-1);
65  printLoad= printLoad(1:NumberLinks);
66  plot(printLoad)
67  axis([1 NumberLinks 0 1])
68  title('Link Loads')
```

2.4 Alínea e)

Para a resolução desta alínea, procedemos à análise e respetiva utilização (com algumas adaptações) do código fornecido nos slides teóricos *"Traffic Engineering in Packet Switched Networks"*.

```
1 % Develop a multi-start local search optimization algorithm for n cycles,
    where n is an
```

```
% input parameter, to find a solution with the lowest network average round-
    trip delay

n = [ 3 10 30 300 1000];

for q=1:length(n)
    GlobalBest= Inf;
    Iterations = n(q);
    for iter=1:Iterations

        CurrentSolution= GreedyRandomized();
        CurrentObjective= EvaluateAverageDelay(CurrentSolution);

        repeat= true;
        while repeat
            NeighbourBest= Inf;

            % evaluate the best neighbour
            for i=1:size(CurrentSolution,1)
                NeighbourSolution= BuildNeighbour(CurrentSolution,i);
                NeighbourObjective= EvaluateAverageDelay(NeighbourSolution);
                if NeighbourObjective < NeighbourBest
                    NeighbourBest= NeighbourObjective;
                    NeighbourBestSolution= NeighbourSolution;
                end
            end

            % evalute if the current is better that the best solution so far
            if NeighbourBest < CurrentObjective
                CurrentObjective= NeighbourBest;
                CurrentSolution= NeighbourBestSolution;
            else
                repeat= false;
            end
        end

        % evaluate the current objective against the global best
        if CurrentObjective < GlobalBest
            GlobalBestSolution= CurrentSolution;
            GlobalBest= CurrentObjective;
        end
    end
    fprintf('Iterations: %d\t GlobalBest: %0.8f\n', Iterations, GlobalBest);
    fprintf('--Max load %0.2f\n\n',EvaluateLoad(GlobalBestSolution));
end
```

Seguidamente procedemos à implementação dos métodos:

- `GreedyRandomized()`

- `EvaluateAverageDelay(CurrentSolution)`
- `BuildNeighbour(CurrentSolution,i)`

```
function solution = GreedyRandomized()

Matrizes;
miu = R*1e9/(8*1000);              % capacidade em bits / pacotes de 1000bytes
    -> pacotes/sec
NumberLinks = sum(sum(R>0));       % numero total de ligacoes
lambda_s = T*1e6/(8*1000);         % packet arrival rate
gama = sum(sum(lambda_s));         % trafego total na rede (packets/sec)
d = L*1e3/2e8;                     % velocidade propagacao / vel da luz na
   fibra optica
pairs = [];

% descobrir nos ligados por um path bidirecional
for origin=1:16
    for destination=(origin+1):17
        if T(origin,destination)+T(destination,origin)>0
            pairs = [pairs; origin destination];
        end
    end
end

npairs = size(pairs,1);
b = randperm(npairs);        % random sorted numbers up to npairs
aux = [];

% reorder pairs acording to b
for i = 1: npairs
    aux(i,:) = pairs(b(i), :);
end

pairs = aux;
lambda = zeros(17);
routes = zeros(npairs,17);

% compute the new lambda between the routes between nodes according the
   shortestpaths
for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);

    delay = 1./(miu-lambda) + d; % M/M/1 Atraso m dio no sistema Modulo 2
   slides
    r = ShortestPathSym(delay, origin, destination);

    routes(i,:) = r;
```

```
42
43      j= 1;
44      while r(j) ~= destination
45          lambda(r(j), r(j+1)) = lambda(r(j), r(j+1)) + lambda_s(origin,
        destination);
46          lambda(r(j+1), r(j)) = lambda(r(j+1), r(j)) + lambda_s(destination,
        origin);
47          j = j + 1;
48      end
49  end
50
51
52  solution = struct('pairs', pairs, 'routes', routes, 'lambda',lambda);
53
54  end
```

```
1   function AverageDelay = EvaluateAverageDelay(solution)
2
3   %pairs = neighbourSolution.pairs;
4   %routes = neighbourSolution.routes;
5   lambda = solution.lambda;
6
7   Matrizes;
8   miu= R*1e9/(8*1000);            % capacidade em bits / pacotes de 1000bytes
        -> pacotes/sec
9   lambda_s= T*1e6/(8*1000);       % packet arrival rate
10  gama= sum(sum(lambda_s));       % trafego total na rede (packets/sec)
11  d= L*1e3/2e8;                   % velocidade propagacao / vel da luz na
        fibra optica
12
13  AverageDelay= (lambda./(miu-lambda)+lambda.*d);
14  AverageDelay(isnan(AverageDelay))= 0;
15  AverageDelay= 2*sum(sum(AverageDelay))/gama;
16
17  end
```

```
1   function solution = BuildNeighbour(solution, i)
2
3   Matrizes;
4   lambda_s = T * 1e6 / (8*1000); % pck arrival rate
5   miu = R * 1e9 / (8*1000);
6   d = L * 1e3 / 2e8;
7
8   origin = solution.pairs(i,1);
9   destination = solution.pairs(i,2);
10  r = solution.routes(i,:);
11
12  % compute the lambda(pckts/s) from our origin up to the destination
13  j = 1;
```

```
while r(j) ~= destination
    solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) - lambda_s(origin,destination);
    solution.lambda(r(j+1),r(j)) = solution.lambda(r(j+1),r(j)) - lambda_s(destination,origin);
    j= j+1;
end

delay = (1./(miu - solution.lambda) + d);           % delay as a metric
r = ShortestPathSym(delay, origin, destination);
solution.routes(i,:) = r;

j = 1;
% recompute the lambda
while r(j) ~= destination
    solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) + lambda_s(origin,destination);
    solution.lambda(r(j+1),r(j))= solution.lambda(r(j+1),r(j)) + lambda_s(destination,origin);
    j= j+1;
end

end
```

2.5 Alínea f)

```
% lowest maximum connection load

n = [ 3 10 30 300 1000];

for q=1:length(n)
    GlobalBest= Inf;
    HighestAverage = 0;
    Iterations = n(q);
    for iter=1:Iterations

        CurrentSolution= GreedyRandomizedLoad();
        CurrentObjective= EvaluateLoad(CurrentSolution);

        repeat= true;
        while repeat
            NeighbourBest= Inf;

            % evaluate the best neighbour
            for i=1:size(CurrentSolution.pairs,1) % size de um struct?
                NeighbourSolution= BuildNeighbourLoad(CurrentSolution,i);
                NeighbourObjective= EvaluateLoad(NeighbourSolution);
```

```
                if NeighbourObjective < NeighbourBest
                    NeighbourBest= NeighbourObjective;
                    NeighbourBestSolution= NeighbourSolution;
                end
            end

            % evalute if the current is better that the best solution so far
            if NeighbourBest < CurrentObjective
                CurrentObjective= NeighbourBest;
                CurrentSolution= NeighbourBestSolution;
            else
                repeat= false;
            end
        end

        % evaluate the current objective against the global best
        if CurrentObjective < GlobalBest
            GlobalBestSolution= CurrentSolution;
            GlobalBest= CurrentObjective;

             if HighestAverage < EvaluateAverageDelay(GlobalBestSolution)
                 HighestAverage = EvaluateAverageDelay(GlobalBestSolution);
             end

        end
    end
    fprintf('Iterations: %d\t GlobalBest: %0.8f\n', Iterations, GlobalBest);
    fprintf('--Average delay %0.8f\n',EvaluateAverageDelay(
   GlobalBestSolution));
    fprintf('--Highest avg delay %0.8f\n\n', HighestAverage);
end
```

Seguidamente procedemos à implementação dos métodos:

- `GreedyRandomizedLoad()`
- `EvaluateLoad(CurrentSolution)`
- `BuildNeighbourLoad(CurrentSolution,i)`

```
function solution = GreedyRandomizedLoad()

(...)

% compute the new lambda between the routes between nodes according the
    shortestpaths
for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
```

```
9
10      Load= lambda./miu;
11      r = ShortestPathSym(Load.^2, origin, destination);
12
13      routes(i,:) = r;
14
15      j= 1;
16      while r(j) ~= destination
17          lambda(r(j), r(j+1)) = lambda(r(j), r(j+1)) + lambda_s(origin,
    destination);
18          lambda(r(j+1), r(j)) = lambda(r(j+1), r(j)) + lambda_s(destination,
    origin);
19          j = j + 1;
20      end
21  end
22
23
24  solution = struct('pairs', pairs, 'routes', routes, 'lambda',lambda);
25
26  end
```

```
1   function metric = EvaluateLoad(solution)
2
3   lambda = solution.lambda;
4
5   Matrizes;
6   miu= R*1e9/(8*1000);
7
8   NumberLinks= sum(sum(R>0));
9
10  Load= lambda./miu;
11  Load(isnan(Load))= 0;
12  %MinumunLoad= min(min(Load));
13  %AverageLoad= sum(sum(Load))/NumberLinks;
14
15  MaximumLoad= max(max(Load));
16
17  metric = MaximumLoad;
18
19  end
```

```
1   function solution = BuildNeighbourLoad(solution, i)
2
3   (...)
4
5   Load= solution.lambda./miu;
6   r = ShortestPathSym(Load.^2, origin, destination);
7   solution.routes(i,:) = r;
8
```

```
9  j = 1;
10 % recompute the lambda
11 while r(j) ~= destination
12     solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) + lambda_s(
       origin,destination);
13
14 (...)
15
16 end
```

Detalhe das pequenas mudanças nas implementação relativamente à questão *e)* que permitem usar como métrica de melhor solução a menor carga máxima.

3 Resultados e conclusões

3.1 Alínea a)

Após a execução do ficheiro `solution_a.m` obtivemos os seguintes resultados para a rede quando os fluxos são criados pela rota mais curta.

- MaximumLoad = 0.9960
- AverageLoad = 0.3478
- AverageDelay = 0.0032
- MaxAvDelay = 0.0065

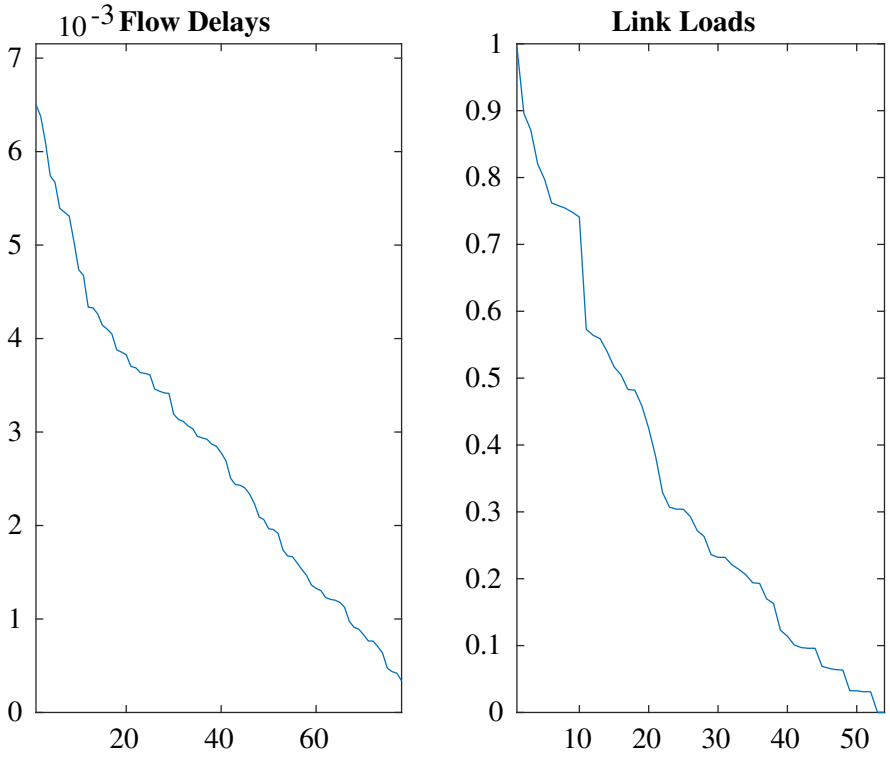

Figura 1: Gráfico para observação do atraso médio em cada fluxo e da carga por cada ligação para a solução A

Podemos observar que nesta estratégia temos conexões que estão praticamente na ocupação máxima enquanto outras estão praticamente desocupadas.

3.2 Alínea b)

3.2.1 Versão 1

Após a execução do ficheiro `solution_b.m` obtivemos os seguintes resultados:

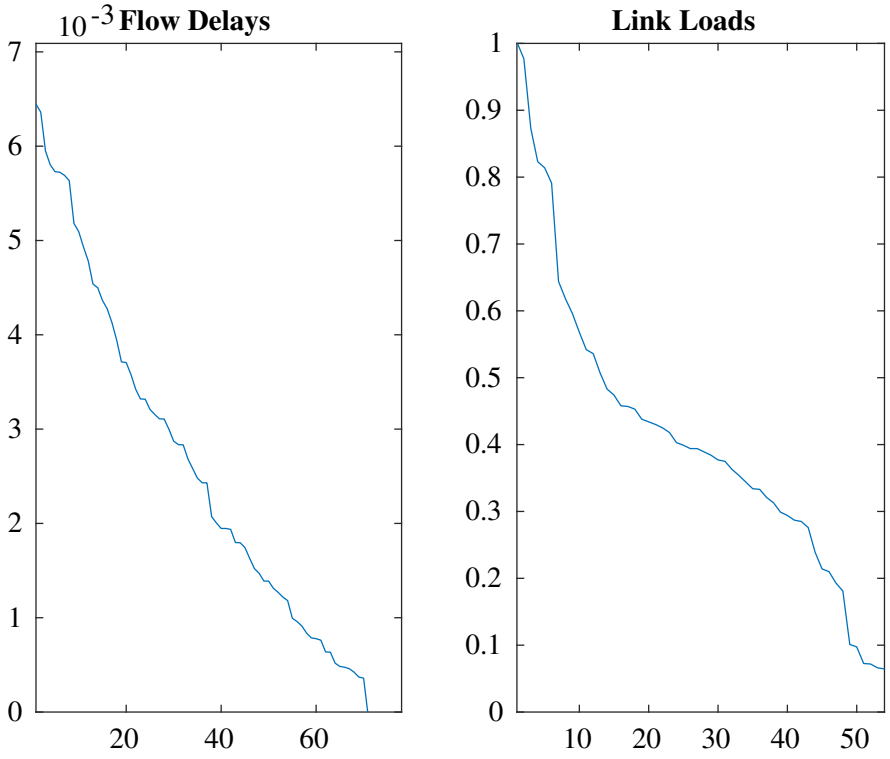

Figura 2: Gráfico para observação do atraso médio em cada fluxo e da carga por cada ligação para a solução B - versão 1

- MaximumLoad = 1.0020 x 10^2 %
- AverageLoad = 0.4109 x 10^2 %
- AverageDelay = 0.0024 x 10^2 %
- MaxAvDelay = 0.0064 x 10^2 %

Visto que como métrica foi utilizado a minimização da carga da ligação podemos ver que existe uma melhor distribuição de carga entre as várias conexões dos nós de rede. Contudo, nesta simulação temos cargas maiores que 100%, o que invalida os resultados, visto que iremos ter conexões em que o *miu* tem contribuição negativa no atraso médio de rede. Este cenário é impossível de acontecer.

3.2.2 Versão 2

Como obtivemos resultados na versão 1 em que o Maximum load é superior a 1 (i.e. >100%), e sendo isso impossível, foi sugerido pelo docente que elevássemos ao quadrado a variável `Load` que é enviada para o método `ShortestPathSym()`, para que esta métrica produzisse resultados mais realísticos. Portanto, esta versão minimiza $Load^2$ em vez de minimizar $Load$.

Após a execução do ficheiro `solution_b_v2.m` obtivemos os seguintes resultados:

- MaximumLoad = 0.7880 x 10^2 %
- AverageLoad = 0.4629 x 10^2 %
- AverageDelay = 0.0037 x 10^2 %
- MaxAvDelay = 0.0089 x 10^2 %

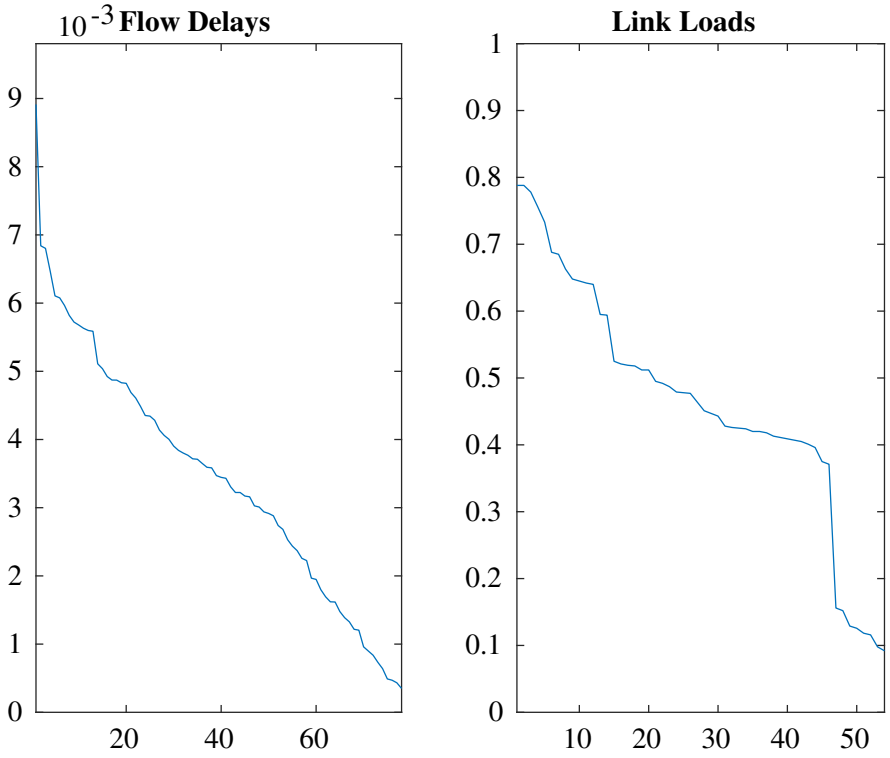

Figura 3: Gráfico para observação do atraso médio em cada fluxo e da carga por cada ligação para a solução B - versão 2

Assim, estes aparentam estar mais coerentes com o que é esperado pela simulação, em que existe uma maior distribuição da carga entre as conexões sem contribuições de atrasos negativos no sistema. A consequência direta desta solução é uma distribuição de carga nas conexões mais homogénea assim como um menor atraso médio do que quando há conexões praticamente saturadas para a primeira simulação.

3.3 Alínea c)

Após a execução do ficheiro `solution_c.m` obtivemos os seguintes resultados:

- MaximumLoad = 0.9100 x 10^2 %
- AverageLoad = 0.3447 x 10^2 %
- AverageDelay = 0.0026 x 10^2 %
- MaxAvDelay = 0.0053 x 10^2 %

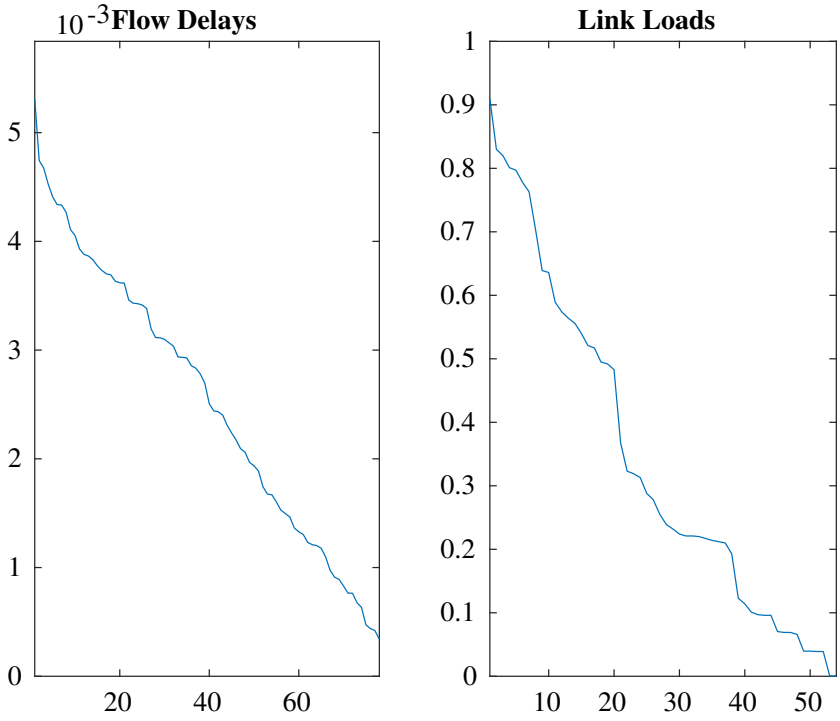

Figura 4: Gráfico para observação do atraso médio em cada fluxo e da carga por cada ligação para a solução C

3.4 Alínea d)

	MaximumLoad	AverageLoad	AverageDelay	MaxAvDelay
Solution_A	0.9960	0.3478	0.0032	0.0065
Solution_B_V1	1.0020	0.4109	0.0024	0.0064
Solution_B_v2	0.7880	0.4629	0.0037	0.0089
Solution_C	0.9100	0.3447	0.0026	0.0053

Tabela 1: Comparação entre resultados obtidos para as diferentes soluções. Valores no intervalo [0,1]

Conclusões: Podemos observar que cada uma das soluções tenta minimizar aquilo a que se propõe, sendo que no caso da solução A o número de saltos, a solução B a carga média e a solução C o atraso médio.

Dos resultados obtidos, podemos classificar as diferentes soluções em duas categorias: a melhor para a experiência do cliente e a melhor para o operador. Quanto menor for a carga melhor para o operador, pois permite ter uma rede com uma ocupação menor e menos propicia à sobrecarga ou à falha. Para o cliente quanto menor for o atraso melhor será a experiência.

Contudo, nenhuma das soluções é equilibrada, visto que uma métrica sacrifica sempre a outra e que em nenhuma existe um equilíbrio entre o que é melhor para os dois lados, isto é, quando se tenta fazer a otimização do atraso médio inevitavelmente há conexões sobrecarregadas e quando se tenta otimizar as cargas a latência aumenta excessivamente.

Fica portanto por estudar uma forma de conseguir balancear as duas métricas numa solução que seja ótima nas duas métricas, ou que pelo menos seja mais equilibrada.

3.5 Alínea e)

Após a execução do script obtivemos os seguintes resultados para esta alínea:

	n=3	**n=10**	**n=30**	**n=300**	**n=1000**
AverageDelay	0.00264106	0.00263961	0.00263913	0.00263907	0.00262524
MaxLoad	0.91	0.91	0.91	0.91	0.85

Tabela 2: Resultados obtidos para alinea e)

Conclusões: Podemos verificar que embora o atraso médio seja efetivamente minimizado, a carga máxima está perto dos limite das conexões. Podemos observar também que o numero de iterações tem pouco impacto na melhor solução, sendo que o valor de *average delay* tende para 0.26%. Justificações possíveis para isto poderá ser a existência de múltiplas soluções que resolvem para um mínimo possível.

3.6 Alínea f)

Após a execução do script obtivemos os seguintes resultados para esta alínea:

	n=3	**n=10**	**n=30**	**n=300**	**n=1000**
MaximumLoad	0.66300000	0.64700000	0.64300000	0.64300000	0.64300000
AverageDelay	0.00339420	0.00346237	0.00325201	0.00366373	0.00372758
Highest average delay	0.00339420	0.00346237	0.00332272	0.00366373	0.00372758

Tabela 3: Resultados obtidos para alinea f)

Conclusões: Neste caso podemos observar que efetivamente a carga máxima é otimizada, mas neste caso sem sacrificar (ao contrário da alínea *e)*) o atraso médio. O número de iterações também parece ter pouca influência em encontrar um melhor resultado, uma vez que a partir das 300 iterações já se observa um valor constante para a caga máxima.

3.7 Alínea g)

Conclusões: Na nossa opinião, visto que a carga dos servidores se relaciona com o atraso médio cremos que a solução que o ISP deve adotar é a da minimização da carga média da rede. Assim, além do impacto no atraso médio ser pouco significativo permite também uma maior tolerância a variações de picos tráfego enquanto mantém valores não muito piores do atraso médio.

Além destas duas vantagem tem também a vantagem de ter uma rede balanceada, que muito provavelmente será mais resistente no caso na falha de um nó, visto que se a ocupação média é menor a rede conseguirá suportar melhor a distribuição da carga do nó que falha.

4 Referências

- Guia prático disponível na página elearning da disciplina
- Slide teóricos disponível na página elearning da disciplina
- Documentação matlab `https://www.mathworks.com/help/matlab/`