Relatório

Assignment Guide No. 3

TRAFFIC ENGINEERING OF PACKET SWITCHED NETWORKS

Desempenho e Dimensionamento de Redes

2014/2015

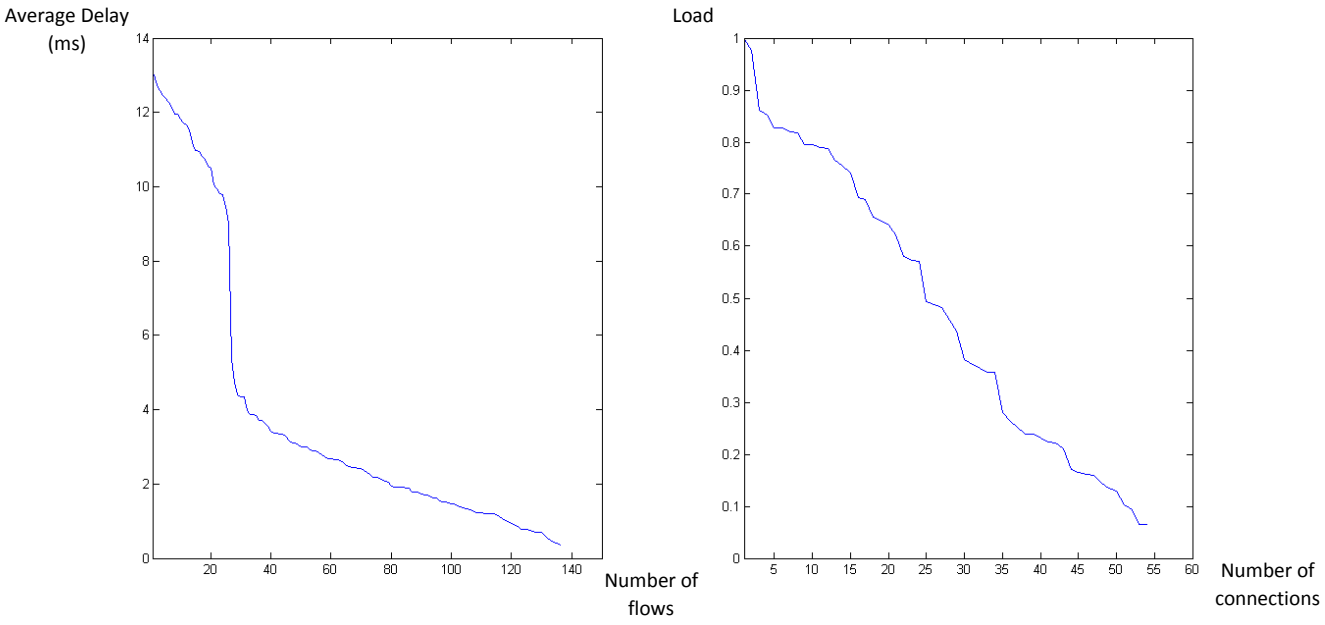
Turma P3
Grupo 7

João Guerra 59290
Mário Pina 65292

Solução A

```
function [averageLoad,maxLoad]= SolutionA()

Matrizes;

routes = [];
load = zeros(17);

for i=1:16
   for j=i+1:17
       route = ShortestPathSym(L, i, j);
       routes = [routes; i, j, route];
       averageTrafficGo = T(i,j);
       averageTrafficCome = T(j,i);
       for k=1:16
            if route(k+1) == 0
                break;
            end

            load(route(k), route(k+1)) = load(route(k), route(k+1)) +
averageTrafficGo;
            load(route(k+1), route(k)) = load(route(k+1), route(k)) +
averageTrafficCome;
       end
   end
end

load = load / 1000;

maxLoad = max(max(load));
averageLoad = sum(sum(load))/sum(sum(R));


lambdas = (load*(10^9)) / (1000*8);

mew = 10^9/8000;

D = L*1000/(3*10^8);

W = lambdas ./ (mew-lambdas) + lambdas.*D;

gama = sum(sum(T))*10^6/8000;

W = 2*sum(sum(W))/gama*1000;

numeroroutes = size(routes,1);

averageRoundTripDelay = [];

for caminho= 1:numeroroutes
    somatoriointermedio = 0;
    i = routes(caminho,1);
    j = routes(caminho,2);
    for k=1:16
       if routes(caminho, 2+k+1) == 0
           break;
       end
       somatoriointermedio = somatoriointermedio + (1/(mew -
(lambdas(routes(caminho, 2+k), routes(caminho,
```

```
2+k+1))))+2*D(routes(caminho, 2+k), routes(caminho, 2+k+1)) + (1/(mew
- (lambdas(routes(caminho, 2+k+1), routes(caminho, 2+k))));
    end
    averageRoundTripDelay = [averageRoundTripDelay; i, j,
somatoriointermedio*1000];
end

averageRoundTripDelay = sortrows(averageRoundTripDelay, -3)

load_plot = sortrows(load(:), -1);
load_plot = load_plot(1:54)
subplot(1,2,1)
plot(averageRoundTripDelay(:,3))
axis([1,150, 0, 14])
subplot(1,2,2)
plot(load_plot)
axis([1,60, 0, 1])
```



Explicação do código:

A solução A começa por calcular o caminho mais curto (com base na distância entre os nós) entre cada 2 pares de nós e a carga na ligação entre cada conexão entre esses 2 nós.

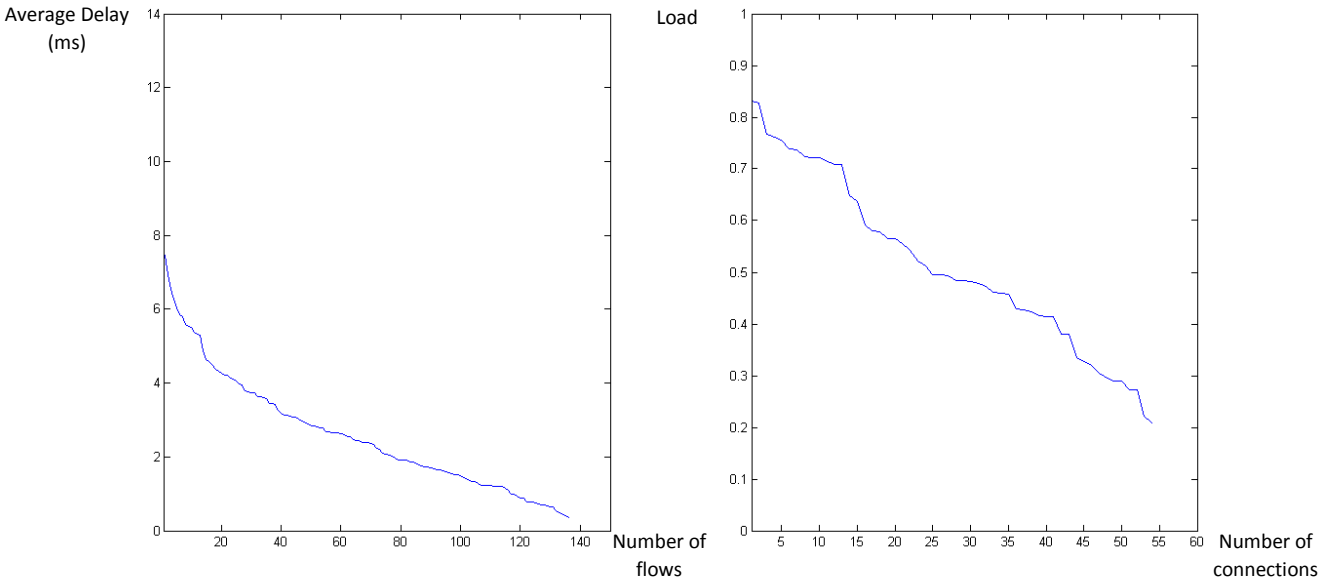
Em seguida utiliza a aproximação de Kleinrock para determinar o atraso médio de transmissão de cada fluxo. Com esses valores é possível calcular facilmente o valor máximo de atraso e o seu fluxo correspondente.

Em seguida foram criados os gráficos (demonstrados em cima) para se verificar o atraso máximo em cada fluxo e a carga por cada ligação.

Solução B

```
function [averageLoad,maxLoad]= SolutionB()
Matrizes;

routes = [];
load = zeros(17);

for i=1:16
   for j=i+1:17
       route = ShortestPathSym(load, i, j);
       routes = [routes; i, j, route];
       averageTrafficGo = T(i,j);
       averageTrafficCome = T(j,i);
       for k=1:16
            if route(k+1) == 0
                break;
            end

            load(route(k), route(k+1)) = load(route(k), route(k+1)) +
averageTrafficGo;
            load(route(k+1), route(k)) = load(route(k+1), route(k)) +
averageTrafficCome;
       end
   end
end

load = load / 1000;

maxLoad = max(max(load));
averageLoad = sum(sum(load))/sum(sum(R));


lambdas = (load*(10^9)) / (1000*8);

mew = 10^9/8000;

D = L*1000/(3*10^8);

W = lambdas ./ (mew-lambdas) + lambdas.*D;

gama = sum(sum(T))*10^6/8000;

W = 2*sum(sum(W))/gama*1000;

numeroroutes = size(routes,1);

averageRoundTripDelay = [];

for caminho= 1:numeroroutes
    somatoriointermedio = 0;
    i = routes(caminho,1);
    j = routes(caminho,2);
    for k=1:16
       if routes(caminho, 2+k+1) == 0
           break;
       end
       somatoriointermedio = somatoriointermedio + (1/(mew -
(lambdas(routes(caminho, 2+k), routes(caminho,
2+k+1))))))+2*D(routes(caminho, 2+k), routes(caminho, 2+k+1)) + (1/(mew
- (lambdas(routes(caminho, 2+k+1), routes(caminho, 2+k)))));
```

```
    end
    averageRoundTripDelay = [averageRoundTripDelay; i, j,
somatoriointermedio*1000];
end

averageRoundTripDelay = sortrows(averageRoundTripDelay, -3)

load_plot = sortrows(load(:), -1);
load_plot = load_plot(1:54)
subplot(1,2,1)

plot(averageRoundTripDelay(:,3))
axis([1,150, 0, 14])
subplot(1,2,2)
plot(load_plot)
axis([1,60, 0, 1])

end
```



Explicação do código:

A solução B começa por calcular o caminho que origina o valor mais baixo da carga entre cada 2 pares de nós e a sua respectiva carga de ligação.

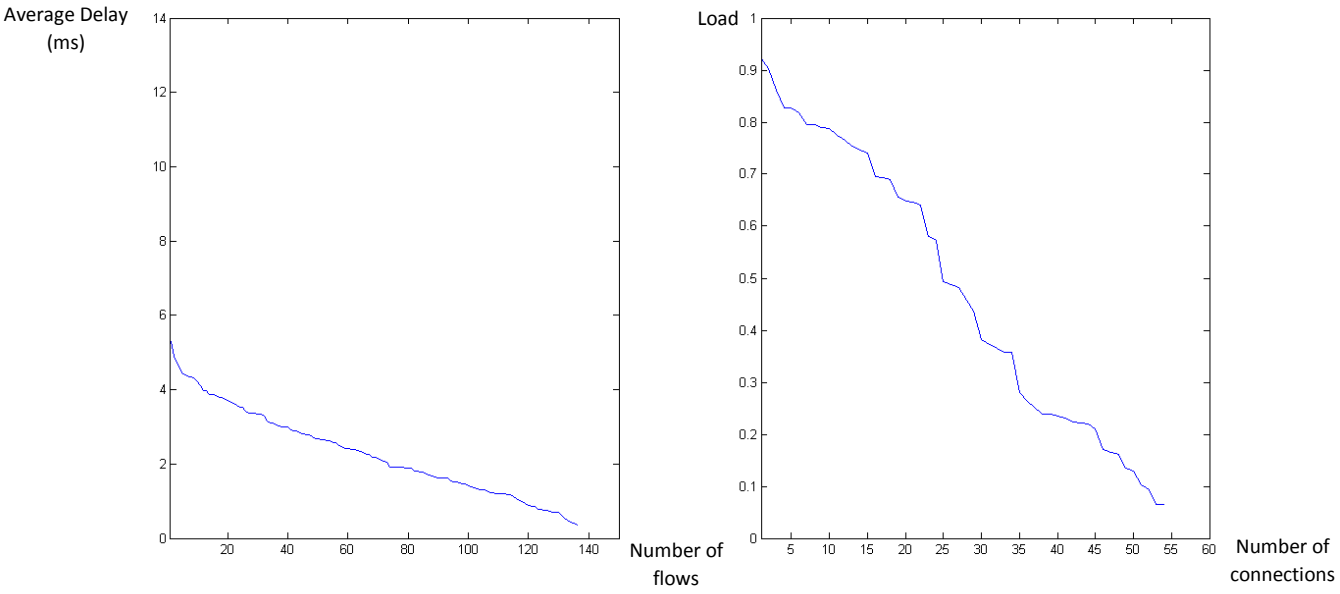
Em seguida utiliza a aproximação de Kleinrock para determinar o atraso médio de transmissão de cada fluxo. Com esses valores é possível calcular facilmente o valor máximo de atraso e o seu fluxo correspondente.

Em seguida foram criados os gráficos (demonstrados em cima) para se verificar o atraso máximo em cada fluxo e a carga por cada ligação.

Solução C

```
function [averageLoad,maxLoad]= SolutionC()
Matrizes;

routes = [];
load = zeros(17);
mew = 10^9/8000;

D = L*1000/(3*10^8);

for i=1:16
   for j=i+1:17
       lambdas = (load*(10^6)) / (1000*8);
       costs = 1./(mew-lambdas) + D;
       route = ShortestPathSym(costs, i, j);
       routes = [routes; i, j, route];
       averageTrafficGo = T(i,j);
       averageTrafficCome = T(j,i);
       for k=1:16
            if route(k+1) == 0
                break;
            end

            load(route(k), route(k+1)) = load(route(k), route(k+1)) +
averageTrafficGo;
            load(route(k+1), route(k)) = load(route(k+1), route(k)) +
averageTrafficCome;
       end
   end
end

load = load / 1000;

maxLoad = max(max(load));
averageLoad = sum(sum(load))/sum(sum(R));


lambdas = (load*(10^9)) / (1000*8);

W = lambdas ./ (mew-lambdas) + lambdas.*D;

gama = sum(sum(T))*10^6/8000;

W = 2*sum(sum(W))/gama*1000;

numeroroutes = size(routes,1);

averageRoundTripDelay = [];

for caminho= 1:numeroroutes
    somatoriointermedio = 0;
    i = routes(caminho,1);
    j = routes(caminho,2);
    for k=1:16
       if routes(caminho, 2+k+1) == 0
           break;
       end
       somatoriointermedio = somatoriointermedio + (1/(mew -
(lambdas(routes(caminho, 2+k), routes(caminho,
```

```
2+k+1))))+2*D(routes(caminho, 2+k), routes(caminho, 2+k+1)) + (1/(mew
- (lambdas(routes(caminho, 2+k+1), routes(caminho, 2+k))));
    end
    averageRoundTripDelay = [averageRoundTripDelay; i, j,
somatoriointermedio*1000];
end

averageRoundTripDelay = sortrows(averageRoundTripDelay, -3)

load_plot = sortrows(load(:), -1);
load_plot = load_plot(1:54)
subplot(1,2,1)
plot(averageRoundTripDelay(:,3))
axis([1,150, 0, 14])
subplot(1,2,2)
plot(load_plot)
axis([1,60, 0, 1])
end
```



Explicação do código:

A solução C começa por calcular o caminho que origina o valor mais baixo de atraso entre cada 2 pares de nós e a sua respectiva carga de ligação.

Em seguida utiliza a aproximação de Kleinrock para determinar o atraso médio de transmissão de cada fluxo. Com esses valores é possível calcular facilmente o valor máximo de atraso e o seu fluxo correspondente.

Em seguida foram criados os gráficos (demonstrados em cima) para se verificar o atraso máximo em cada fluxo e a carga por cada ligação.

Comparação de soluções

Como podemos verificar a solução A é aquela que têm um número maior para a carga máxima e atraso máximo entre as ligações, uma vez que a única coisa que tem em consideração é a distância entre os nós, não fazendo nenhuma optimização para reduzir nenhum dos restantes campos.

A solução B tem uma carga máxima entre as ligação mais baixa do que as outras soluções, pois tenta minimizar esse campo no cálculo do melhor caminho para cada fluxo, enquanto a solução C é aquela com um atraso máximo de cada fluxo mais baixo.

Assim a solução B será a mais propícia para o lado dos servidores, uma vez que minimiza o processamento em cada 1 dos nós, evitando mais facilmente perda de pacotes e sobrecarga do sistema. A solução C é melhor para os clientes, uma vez que minimiza os atrasos no sistema, fazendo com que as ligações entre nós sejam mais rápidas, diminuindo o tempo de acesso à informação.

Optimize Average Round-trip Delay

```
function [F_MaxDelay F_AverageDelay F_MaxLoad]= Optimize(Iterations)
Matrizes;
GlobalBest= Inf;
for iter=1:Iterations
    routes= BuildSolution();
    [CurMaxDelay CurAverageDelay CurMaxLoad]= Evaluate(routes);
    repeat= true;
    while repeat
        LocalBest= Inf;
        for i=1:size(routes,1)
            routesAux= BuildNeighbor(routes,i);
            [MaxDelay AverageDelay MaxLoad]= Evaluate(routesAux);
            if AverageDelay < LocalBest
                LocalBest= AverageDelay;
                LocalBestRoutes= routesAux;
            end
        end
        if LocalBest < CurAverageDelay
            CurAverageDelay= LocalBest;
            routes= LocalBestRoutes;
        else
            repeat= false;
        end
    end
    if LocalBest < GlobalBest
        GlobalBest= LocalBest;
        [F_MaxDelay F_AverageDelay F_MaxLoad]= Evaluate(routes);
        GlobalBestRoutes= routes;
    end
end
end

function routes = BuildSolution()
Matrizes;
ord = [];
for i=1:16
    for j=(i+1):17
        ord= [ord; i j];
    end
end
npares= size(ord,1);
b= randperm(npares);
for i= 1:npares
    ordem(i,:)= ord(b(i),:);
end

load = zeros(17);
mew = 10^9/8000;
D = L*1000/(3*10^8);
routes = [];

for i=1:size(ordem,1)
    lambdas = (load*(10^6)) / (1000*8);
    costs = 1./(mew-lambdas) + D;
    route = ShortestPathSym(costs, ordem(i,1), ordem(i,2));
    routes = [routes; ordem(i,1), ordem(i,2), route];
    averageTrafficGo = T(ordem(i,1), ordem(i,2));
    averageTrafficCome = T(ordem(i,2), ordem(i,1));
    for k=1:16
```

```
        if route(k+1) == 0
            break;
        end
        load(route(k), route(k+1)) = load(route(k), route(k+1)) +
averageTrafficGo;
        load(route(k+1), route(k)) = load(route(k+1), route(k)) +
averageTrafficCome;
    end
end

end

function [MD AD ML]= Evaluate(routes)
Matrizes;
load = zeros(17);
mew = 10^9/8000;
D = L*1000/(3*10^8);

for i=1:size(routes,1)
    averageTrafficGo = T(routes(i,1), routes(i,2));
    averageTrafficCome = T(routes(i,2), routes(i,1));
    for k=1:16
        if routes(i,k+3) == 0
            break;
        end
        load(routes(i,k+2), routes(i,k+3)) = load(routes(i,k+2),
routes(i,k+3)) + averageTrafficGo;
        load(routes(i,k+3), routes(i,k+2)) = load(routes(i,k+3),
routes(i,k+2)) + averageTrafficCome;
    end
end

load = load / 1000;
mew = 10^9/8000;
D = L*1000/(3*10^8);
ML = max(max(load));
lambdas = (load*(10^9)) / (1000*8);
averageRoundTripDelay = [];

for i=1:size(routes,1)
    somatoriointermedio = 0;
    inicio = routes(i,1);
    fim = routes(i,2);
    for j=1:size(routes,2) -2
        if routes(i, 2+j+1) == 0
           break;
        end
       somatoriointermedio = somatoriointermedio + (1/(mew -
(lambdas(routes(i, 2+j), routes(i, 2+j+1))))))+2*D(routes(i, 2+j),
routes(i, 2+j+1)) + (1/(mew - (lambdas(routes(i, 2+j+1), routes(i,
2+j))))));
    end
    averageRoundTripDelay = [averageRoundTripDelay; inicio, fim,
somatoriointermedio*1000];
end
averageRoundTripDelay = sortrows(averageRoundTripDelay, -3);
MD = averageRoundTripDelay(1, 3);

W = lambdas ./ (mew-lambdas) + lambdas.*D;

gama = sum(sum(T))*10^6/8000;
```

```
AD = 2*sum(sum(W))/gama*1000; % milisegundos

end

function routesAux = BuildNeighbor(routes,j)
Matrizes;
load = zeros(17);
mew = 10^9/8000;
D = L*1000/(3*10^8);

for i=1:size(routes,1)
    if i ~= j
        averageTrafficGo = T(routes(i,1), routes(i,2));
        averageTrafficCome = T(routes(i,2), routes(i,1));
        for k=1:16
            if routes(i,k+3) == 0
                break;
            end
            load(routes(i,k+2), routes(i,k+3)) = load(routes(i,k+2),
routes(i,k+3)) + averageTrafficGo;
            load(routes(i,k+3), routes(i,k+2)) = load(routes(i,k+3),
routes(i,k+2)) + averageTrafficCome;
        end
    end
end

routesAux = routes;

lambdas = (load*(10^6)) / (1000*8);
costs = 1./(mew-lambdas) + D;
route = ShortestPathSym(costs, routes(j,1), routes(j,2));
routesAux(j, :) = [routes(j,1), routes(j,2), route];
averageTrafficGo = T(routesAux(i,1), routesAux(j,2));
averageTrafficCome = T(routesAux(j,2), routesAux(j,1));
for k=1:16
    if route(k+1) == 0
        break;
    end
    load(routesAux(j,k+2), routesAux(j,k+3)) = load(routesAux(j,k+2),
routesAux(j,k+3)) + averageTrafficGo;
    load(routesAux(j,k+3), routesAux(j,k+2)) = load(routesAux(j,k+3),
routesAux(j,k+2)) + averageTrafficCome;
end
end
```

Number of Times	Maximum Delay	Average Delay	Maximum Load
3	5.3733	2.4183	0.8720
10	5.3749	2.4170	0.8730
30	5.3826	2.4117	0.8130
100	5.3826	2.4120	0.8220

Explicação do código:

Este código tem por base o código fornecido no anexo II do guião prático 3 da disciplina, a função optimize foi copiada do documento e foram implementadas as funções BuildSolution, Evaluate e BuildNeighbor.

Para a função BuildSolution foi inicialmente criada uma matriz representativa da ligação entre cada 2 nós, e ordenada de forma aleatória. Em seguida foi usada essa matriz para calcular o melhor caminho entre cada 2 pares de nós, para que o atraso médio calculado por fluxo fosse o menor possível.

Na função Evaluate foi novamente calculada a carga entre ligações e foram obtidos os valores médios de atraso e os valores máximos de carga e de atraso por fluxo.

A função BuildNeighbor é utilizada para alterar o caminho de um único fluxo de forma a tentar reduzir o atraso médio dos fluxos.

Optimize Maximum Connection Load

```
function [F_MaxDelay F_AverageDelay F_MaxLoad]=
OptimizeLoad(Iterations)
Matrizes;
GlobalBest= Inf;
for iter=1:Iterations
    routes= BuildSolution();
    [CurMaxDelay CurAverageDelay CurMaxLoad]= Evaluate(routes);
    repeat= true;
    while repeat
        LocalBest= Inf;
        for i=1:size(routes,1)
            routesAux= BuildNeighbor(routes,i);
            [MaxDelay AverageDelay MaxLoad]= Evaluate(routesAux);
            if MaxLoad < LocalBest
                LocalBest= MaxLoad;
                LocalBestRoutes= routesAux;
            end
        end
        if LocalBest < CurMaxLoad
            CurMaxLoad= LocalBest;
            routes= LocalBestRoutes;
        else
            repeat= false;
        end
    end
    if LocalBest < GlobalBest
        GlobalBest= LocalBest;
        [F_MaxDelay F_AverageDelay F_MaxLoad]= Evaluate(routes);
        GlobalBestRoutes= routes;
    end
end
end

function routes = BuildSolution()
Matrizes;
ord = [];
for i=1:16
    for j=(i+1):17
        ord= [ord; i j];
    end
end
npares= size(ord,1);
b= randperm(npares);
for i= 1:npares
    ordem(i,:)= ord(b(i),:);
end

load = zeros(17);
mew = 10^9/8000;
D = L*1000/(3*10^8);
routes = [];

for i=1:size(ordem,1)
    route = ShortestPathSym(load, ordem(i,1), ordem(i,2));
    routes = [routes; ordem(i,1), ordem(i,2), route];
    averageTrafficGo = T(ordem(i,1), ordem(i,2));
    averageTrafficCome = T(ordem(i,2), ordem(i,1));
```

```
    for k=1:16
        if route(k+1) == 0
            break;
        end
        load(route(k), route(k+1)) = load(route(k), route(k+1)) +
averageTrafficGo;
        load(route(k+1), route(k)) = load(route(k+1), route(k)) +
averageTrafficCome;
    end
end

end

function [MD AD ML]= Evaluate(routes)
Matrizes;
load = zeros(17);
mew = 10^9/8000;
D = L*1000/(3*10^8);

for i=1:size(routes,1)
    averageTrafficGo = T(routes(i,1), routes(i,2));
    averageTrafficCome = T(routes(i,2), routes(i,1));
    for k=1:16
        if routes(i,k+3) == 0
            break;
        end
        load(routes(i,k+2), routes(i,k+3)) = load(routes(i,k+2),
routes(i,k+3)) + averageTrafficGo;
        load(routes(i,k+3), routes(i,k+2)) = load(routes(i,k+3),
routes(i,k+2)) + averageTrafficCome;
    end
end

load = load / 1000;
mew = 10^9/8000;
D = L*1000/(3*10^8);
ML = max(max(load));
lambdas = (load*(10^9)) / (1000*8);
averageRoundTripDelay = [];

for i=1:size(routes,1)
    somatoriointermedio = 0;
    inicio = routes(i,1);
    fim = routes(i,2);
    for j=1:size(routes,2) -2
        if routes(i, 2+j+1) == 0
           break;
        end
       somatoriointermedio = somatoriointermedio + (1/(mew -
(lambdas(routes(i, 2+j), routes(i, 2+j+1)))))+2*D(routes(i, 2+j),
routes(i, 2+j+1)) + (1/(mew - (lambdas(routes(i, 2+j+1), routes(i,
2+j)))));
    end
    averageRoundTripDelay = [averageRoundTripDelay; inicio, fim,
somatoriointermedio*1000];
end
averageRoundTripDelay = sortrows(averageRoundTripDelay, -3);
MD = averageRoundTripDelay(1, 3);

W = lambdas ./ (mew-lambdas) + lambdas.*D;
```

```
gama = sum(sum(T))*10^6/8000;

AD = 2*sum(sum(W))/gama*1000; % milisegundos

end

function routesAux = BuildNeighbor(routes,j)
Matrizes;
load = zeros(17);
mew = 10^9/8000;
D = L*1000/(3*10^8);

for i=1:size(routes,1)
    if i ~= j
        averageTrafficGo = T(routes(i,1), routes(i,2));
        averageTrafficCome = T(routes(i,2), routes(i,1));
        for k=1:16
            if routes(i,k+3) == 0
                break;
            end
            load(routes(i,k+2), routes(i,k+3)) = load(routes(i,k+2),
routes(i,k+3)) + averageTrafficGo;
            load(routes(i,k+3), routes(i,k+2)) = load(routes(i,k+3),
routes(i,k+2)) + averageTrafficCome;
        end
    end
end

routesAux = routes;

route = ShortestPathSym(load, routes(j,1), routes(j,2));
routesAux(j, :) = [routes(j,1), routes(j,2), route];
averageTrafficGo = T(routesAux(i,1), routesAux(j,2));
averageTrafficCome = T(routesAux(j,2), routesAux(j,1));
for k=1:16
    if route(k+1) == 0
        break;
    end
    load(routesAux(j,k+2), routesAux(j,k+3)) = load(routesAux(j,k+2),
routesAux(j,k+3)) + averageTrafficGo;
    load(routesAux(j,k+3), routesAux(j,k+2)) = load(routesAux(j,k+3),
routesAux(j,k+2)) + averageTrafficCome;
end
end
```

Number of Times	Maximum Delay	Average Delay	Maximum Load
3	5.8412	2.6113	0.7390
10	6.3031	2.6283	0.7320
30	5.7631	2.6833	0.7090
100	5.5996	2.793	0.7070

Explicação do código:

Este código tem por base o código fornecido no anexo II do guião prático 3 da disciplina, a função optimize foi copiada do documento e alterada de forma a calcular a solução com a melhor carga máxima em vez do melhor atraso médio e foram implementadas as funções BuildSolution, Evaluate e BuildNeighbor.

Para a função BuildSolution foi inicialmente criada uma matriz representativa da ligação entre cada 2 nós, e ordenada de forma aleatória. Em seguida foi usada essa matriz para calcular o melhor caminho entre cada 2 pares de nós, para que a carga no caminho percorrido por esse fluxo fosse a menor possível.

Na função Evaluate foi novamente calculada a carga entre ligações e foram obtidos os valores médios de atraso e os valores máximos de carga e de atraso por fluxo.

A função BuildNeighbor é utilizada para alterar o caminho de um único fluxo de forma a tentar reduzir a carga máxima dos fluxos.

Comparação das soluções de optimização:

Na nossa opinião, a solução que optimiza a carga é a melhor solução para o ISP, uma vez que a diferença dos atrasos é na gama das décimas de milissegundo enquanto que a melhoria na carga do sistema é uma ganho de aproximadamente 15%, o que fornece uma maior vantagem na redução da perda dos pacotes e carga geral do sistema.