Desempenho e Dimensionamento de Redes

Report 4

Traffic engineering of packet switched networks

Autores:
João Quintanilha 68065
Bruno Henriques 68314

Professor:
Amaro de Sousa

universidade de aveiro

June 12, 2018

Index

1 Question 2 **2**

1.1 Question 2.a 2
1.2 Question 2.b 3
1.3 Question 2.c 3
1.4 Question 2.d 6
1.5 Question 2.e 6
1.6 Question 2.f 7
1.7 Question 2.g 8
1.8 Question 2.h 14
1.9 Question 2.i 19

1 Question 2

1.1 Question 2.a

Using the script provided by the guide shown in figure 1 we obtained the results:

$$MaximumLoad = 0.99 \text{ and } AvarageLoad = 0.34$$

With the comments on the script figure 1, we try to explain the best of the following script logic throughout the code.

```
Matrizes; %load matrizes
miu= R*1e9/(8*1000); %capacidade convertida
NumberLinks= sum(sum(R>0)); %valor de ligacoes unidirecionais
T(:,3:4)= T(:,3:4)*1e6/(8*1000); %mbs para pacotes por segundo
gama= sum(sum(T(:,3:4)));
d= L*1e3/2e8;   % matrix de atrasos de propagacao
nT= size(T,1);  %nr de pares origem/destino
lambda= zeros(20);  % nr pacotes/s que passa em cada percurso
routes= zeros(nT,20);   % percursos escolhidos


for i=1:nT
    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3); %origem->destino
    lambda_do= T(i,4); %destino->origem
    r= ShortestPathSym(d,origin,destination);
    routes(i,:)= r;
    j= 1;

    %actualiza o lambda para os valores do shortest path
    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
        j= j+1;
    end
end


Load= lambda./miu;
Load(isnan(Load))= 0;
MaximumLoad = max(max(Load))
AverageLoad = sum(sum(Load))/NumberLinks
```

Figure 1: Script in appendix II

1.2 Question 2.b

We created the script shown in figure 2 according to the KLEINROCK APPROXIMATION formulas given on the guide. Running it we got the results for both the network average round-trip delay (W) and the maximum average round-trip time among all flows (Ws):

$$W = 0.003 \quad \text{and} \quad Ws = 5.97 * 10^{-3}$$

```
% 2.b
%(i)
soma = (lambda ./ (miu - lambda)) + lambda .* d;
soma(isnan(soma))= 0;
w = (1/gama) *  sum(sum(soma)) * 2;

%(ii)
ws= zeros(size(routes));
for i=1:nT
    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3); %origem->destino
    lambda_do= T(i,4); %destino->origem
    r= routes(i,:);
    j= 1;

    while r(j)~= destination
        ws(i) = ws(i) + (1 / (miu(r(j),r(j+1)) - lambda(r(j),r(j+1)))) + d(r(j),r(j+1));
        ws(i) = ws(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

max(ws)
```

Figure 2: Exercice 2b

1.3 Question 2.c

For both i) and ii) we created a little script to plot both the average packet round-trip delay of each flow and the load of each direction of each connection showing on the figure 3 the outcome of the code represented on the figure 4

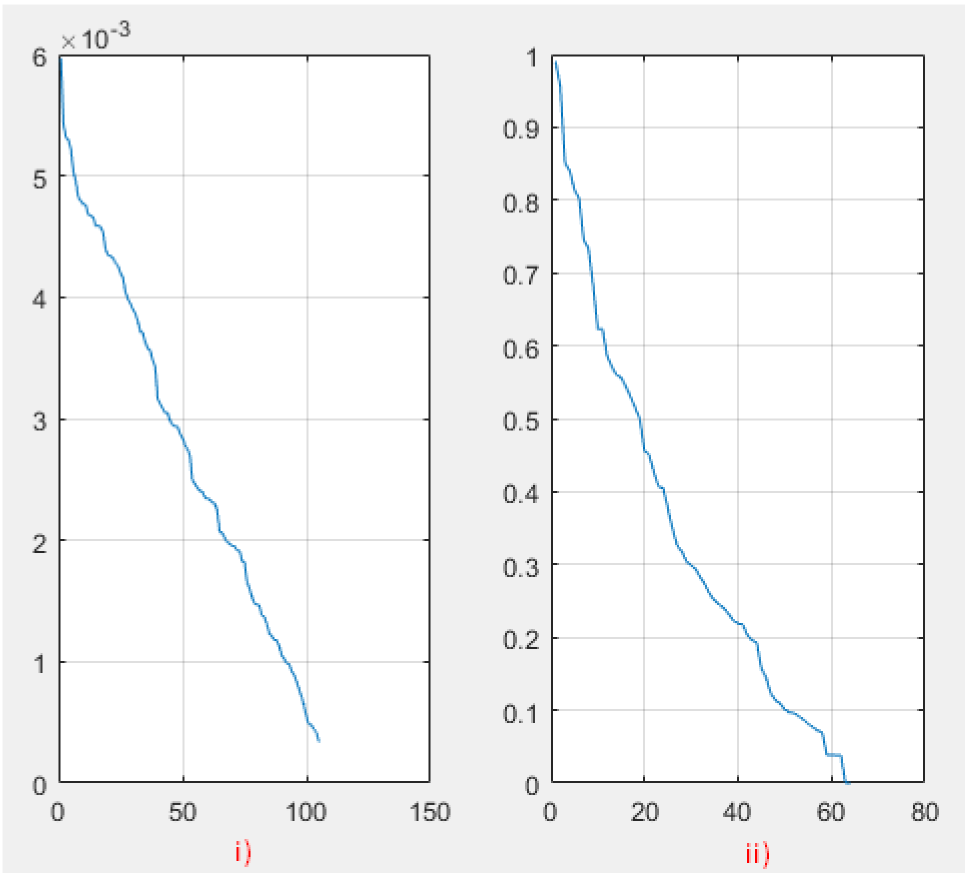

Figure 3: Exercice 2c plots

```
%%%%% Alinea c) %%%%%
%%% (i) %%%
figure(1)
subplot(1,2,1)
ws = sort(ws, 'descend');

plot(ws)
grid on

%%% (ii) %%%
subplot(1,2,2)
a = Load(:);
a = sort(a, 'descend');
a = a(1:NumberLinks);

plot(a)
grid on
```

Figure 4: Exercice 2.c Code

1.4 Question 2.d

For **solution B** using the the previous script with the changes shown in figure 5. With it, for a), we obtained the values:

$$MaximumLoad = 0.737 \quad \text{and} \quad AvarageLoad = 0.379$$

For b), we obtained the values:

$$W = 0.0033 \quad \text{and} \quad Ws = 8 * 10^{-3}$$

And for c) we represent it together with 2.e, in 2.f in figure 7

```
for i=1:nT
    Load= lambda./miu;
    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3); %origem->destino
    lambda_do= T(i,4); %destino->origem
    r= ShortestPathSym(Load,origin,destination);
    routes(i,:)= r;
    j= 1;

    %actualiza o lambda para os valores do shortest path
    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
         j= j+1;
    end


end
```

Figure 5: Solution B

1.5 Question 2.e

For **solution C** using the the previous script with the changes shown in figure 6. With it, for a), we obtained the values:

$$MaximumLoad = 0.89 \quad \text{and} \quad AvarageLoad = 0.34$$

For b), we obtained the values:

$$W = 0.0027 \quad \text{and} \quad Ws = 5.3 * 10^{-3}$$

```
for i=1:nT
    aux = 1./(miu-lambda)+d;
    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3); %origem->destino
    lambda_do= T(i,4); %destino->origem
    r= ShortestPathSym(aux,origin,destination);
    routes(i,:)= r;
    j= 1;

    %actualiza o lambda para os valores do shortest path
    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
         j= j+1;
    end


end
```

Figure 6: Solution C

1.6 Question 2.f

From the results and plots obtained in figure 7, there are some things we can conclude.

If the objective of the solution is to have a faster connection, then solution C would be the best, and solution B being the worst.

Otherwise, if the objective is to have a lower load in the connection then, solution B would be the best, and solution A would be the worst.

With this being said, solution C seems to be the most balanced, since it is the best regarding connection speed and is the second best regarding connection load.

In conclusion, it can be said that the best solution depends on the requirements of the problem.

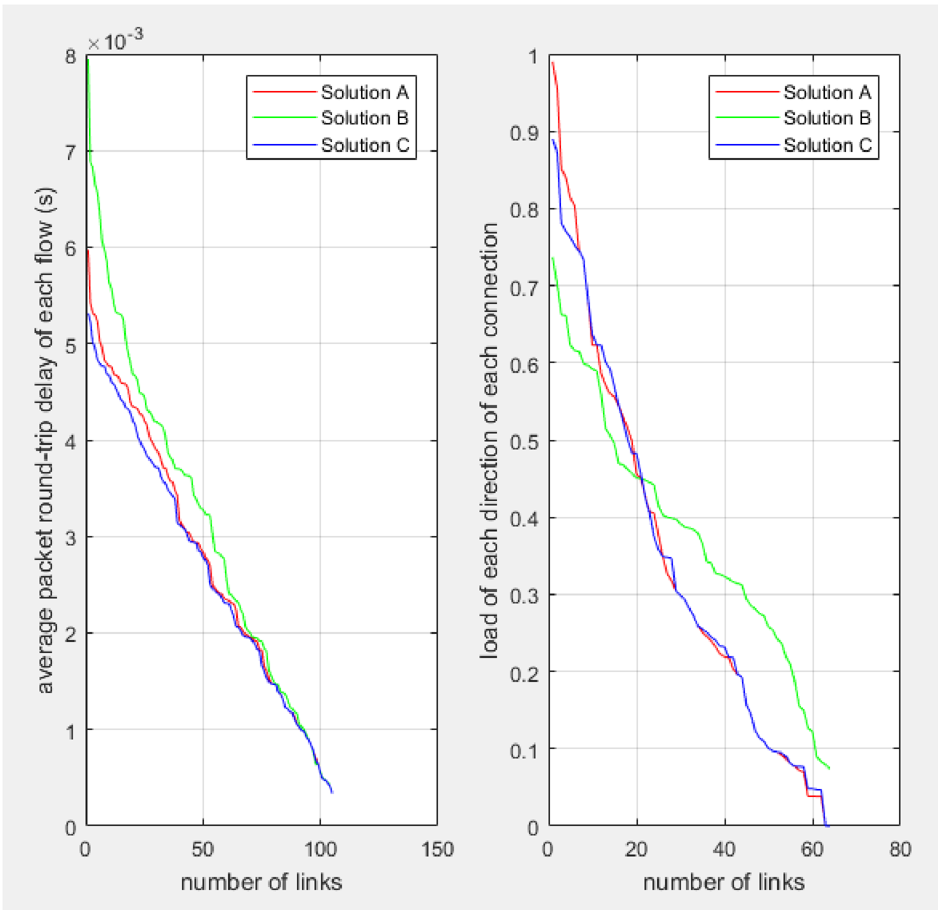

Figure 7: Plot with all solution values

1.7 Question 2.g

Bellow, we show figures for each individual function of our solution. For a more exact result we decided to run the algorithm (figure 11) 500 times.

Showing the results in figure 12. (Note: in figure 11 we paragraphed the code in order to fit the code on the report)

```
function CurrentSolution = GreedyRandomized(nT, miu, d, T)


    lambda = zeros(20);
  for i=randperm(nT)
    aux = 1./(miu - lambda) + d;

    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3);
    lambda_do= T(i,4);
    r= ShortestPathSym(aux,origin,destination);
    routes(i,:)= r;
    j= 1;

    while r(j)~= destination
      lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
      lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
      j= j+1;
    end
  end

  CurrentSolution.routes = routes;
  CurrentSolution.lambda = lambda;
end
```

Figure 8: GreedyRandomized function

```
function CurrentObjective = Evaluate(CurrentSolution, miu, d, gama, nT, T)

  lambda = CurrentSolution.lambda;
  routes = CurrentSolution.routes;

  %%%% b)
  %%% (i) %%%
  soma = ( lambda./ ( miu - lambda ) ) + lambda.* d ;
  soma(isnan(soma)) = 0 ;

  W = (1/gama) * sum(sum(soma)) * 2 ;

  %%% (ii) %%%
  Ws = zeros(nT,1);

  for i=1:nT

    destination= T(i,2);
    r = routes(i,:);
    j = 1;

    while r(j) ~= destination
      Ws(i) = Ws(i) + ((1 / (miu(r(j),r(j+1)) - lambda(r(j),r(j+1)))) + d(r(j),r(j+1)));
      Ws(i) = Ws(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

  CurrentObjective.w = W;
  CurrentObjective.ws = max(Ws);

end
```

Figure 9: Evaluate function

```matlab
function NeighbourSolution = BuildNeighbour(CurrentSolution, i, T, miu, d)

    routes = CurrentSolution.routes;
    lambda = CurrentSolution.lambda;

    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3);
    lambda_do= T(i,4);
    r = routes(i,:);
    j =1;

    while r(j)~= destination
      lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) - lambda_od;
      lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) - lambda_do;
      j= j+1;
    end

    aux = 1./(miu - lambda) + d;
    r= ShortestPathSym(aux,origin,destination);
    routes(i,:)= r;
    j= 1;

    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
        j= j+1;
    end

NeighbourSolution.routes = routes;
NeighbourSolution.lambda = lambda;

end
```

Figure 10: Build neighbor function

```
while counter < 500

    CurrentSolution = GreedyRandomized(nT, miu,d, T);
    CurrentObjective= Evaluate(CurrentSolution, miu, d, gama, nT, T);

    repeat= true;

    while repeat
        NeighbourBest.w = Inf;
        NeighbourBest.ws = Inf;

        for i=1:size(CurrentSolution.routes,1)
            NeighbourSolution= BuildNeighbour(CurrentSolution,i, T, miu, d);
            NeighbourObjective= Evaluate(NeighbourSolution,  miu, d, gama, nT, T);

            if NeighbourObjective.w < NeighbourBest.w || (NeighbourObjective.w
                == NeighbourBest.w && NeighbourObjective.ws < NeighbourBest.ws)
                NeighbourBest = NeighbourObjective;
                NeighbourBestSolution= NeighbourSolution;
            end
        end

        if NeighbourBest.w < CurrentObjective.w || (NeighbourBest.w
            == CurrentObjective.w  &&  NeighbourBest.ws < CurrentObjective.ws)
            CurrentObjective= NeighbourBest;
            CurrentSolution= NeighbourBestSolution;

        else
            repeat= false;
        end
    end

    if CurrentObjective.w < GlobalBest.w || (CurrentObjective.w
        == GlobalBest.w && CurrentObjective.ws < GlobalBest.ws)
        GlobalBestSolution= CurrentSolution;
        GlobalBest= CurrentObjective;
        counter = 0;
    else
        counter = counter +1;
    end

end
```

Figure 11: Solution D

```
>> solutionD
GlobalBest W = 0.002716471
GlobalBest Ws = 0.005315127

A)
MaximumLoad = 0.818000000
AverageLoad = 0.338895312
B.i)
W = 0.002716471
B.ii)
Ws = 0.005315127

C.i)
C.ii)
```

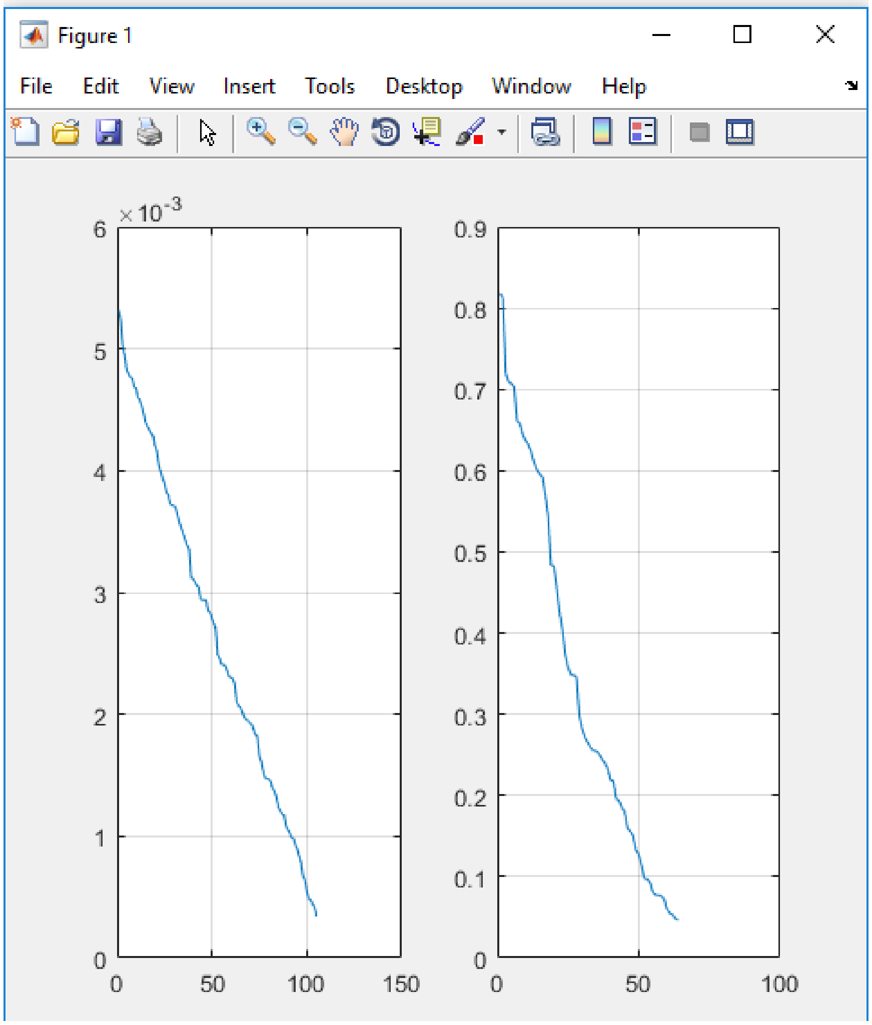

Figure 12: Results from Solution D

1.8 Question 2.h

Adapting the previous algorithm with the objective to find a solution with the lowest maximum connection load and, among all such solutions, the one with the lowest average round-trip delay, bellow we show figures for each individual function of our solution. Again, as in Question 2.g, we ran the algorithm 500 times for a more exact result. We highlighted the changes throughout the code and as for the algorithm itself, adapting the conditions to follow. (Note: in figure 16 we paragraphed the code in order to fit the code on the report)

```
function CurrentSolution = GreedyRandomizedH(nT, miu, d, T)


    lambda = zeros(20);
  for i=randperm(nT)
    Load= lambda./miu;

    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3);
    lambda_do= T(i,4);
    r= ShortestPathSym(Load,origin,destination);
    routes(i,:)= r;
    j= 1;

    while r(j)~= destination
      lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
      lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
      j= j+1;
    end
  end

  CurrentSolution.routes = routes;
  CurrentSolution.lambda = lambda;
end
```

Figure 13: GreedyRandomized function for Solution E

```
function CurrentObjective = EvaluateH(CurrentSolution, miu, d, gama, nT, T)

  lambda = CurrentSolution.lambda;
  routes = CurrentSolution.routes;

  %%%% b)
  %%% (i) %%%
  soma = ( lambda./ ( miu - lambda ) ) + lambda.* d ;
  soma(isnan(soma)) = 0 ;

  W = (1/gama) * sum(sum(soma)) * 2 ;

  %%% (CARGA) %%%

  Load= lambda./miu;
  Load(isnan(Load))= 0;
  MaximumLoad = max(max(Load));

  CurrentObjective.w = W;
  CurrentObjective.MaximumLoad = MaximumLoad;

end
```

Figure 14: Evaluate function for Solution E

```
function NeighbourSolution = BuildNeighbourH(CurrentSolution, i, T, miu, d)

    routes = CurrentSolution.routes;
    lambda = CurrentSolution.lambda;

    origin= T(i,1);
    destination= T(i,2);
    lambda_od= T(i,3);
    lambda_do= T(i,4);
    r = routes(i,:);
    j =1;

    while r(j)~= destination
      lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) - lambda_od;
      lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) - lambda_do;
      j= j+1;
    end

    Load= lambda./miu;
    r= ShortestPathSym(Load,origin,destination);
    routes(i,:)= r;
    j= 1;

    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + lambda_od;
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + lambda_do;
        j= j+1;
    end

NeighbourSolution.routes = routes;
NeighbourSolution.lambda = lambda;

end
```

Figure 15: Build neighbor function for Solution E

```
while counter < 5

    CurrentSolution = GreedyRandomizedH(nT, miu,d, T);
    CurrentObjective= EvaluateH(CurrentSolution, miu, d, gama, nT, T);

    repeat= true;

    while repeat
        NeighbourBest.w = Inf;
        NeighbourBest.MaximumLoad = Inf;

        for i=1:size(CurrentSolution.routes,1)
            NeighbourSolution= BuildNeighbourH(CurrentSolution,i, T, miu, d);
            NeighbourObjective= EvaluateH(NeighbourSolution,  miu, d, gama, nT, T);

            if NeighbourObjective.MaximumLoad < NeighbourBest.MaximumLoad || (NeighbourObjective.MaximumLoad
                == NeighbourBest.MaximumLoad && NeighbourObjective.w < NeighbourBest.w)
                NeighbourBest = NeighbourObjective;
                NeighbourBestSolution= NeighbourSolution;
            end
        end

        if NeighbourBest.MaximumLoad < CurrentObjective.MaximumLoad || (NeighbourBest.MaximumLoad
            == CurrentObjective.MaximumLoad  &&  NeighbourBest.w < CurrentObjective.w)
            CurrentObjective= NeighbourBest;
            CurrentSolution= NeighbourBestSolution;

        else
            repeat= false;
        end
    end

    if CurrentObjective.MaximumLoad < GlobalBest.MaximumLoad || (CurrentObjective.MaximumLoad
        == GlobalBest.MaximumLoad && CurrentObjective.w < GlobalBest.w)
        GlobalBestSolution= CurrentSolution;
        GlobalBest= CurrentObjective;
        counter = 0;
    else
        counter = counter +1;
    end

end
```

Figure 16: Solution E

```
>> SolutionE
GlobalBest W = 0.002906082
GlobalBest MaximumLoad = 0.559000000

A)
MaximumLoad = 0.559000000
AverageLoad = 0.347104688
B.i)
W = 0.002906082
B.ii)
Ws = 0.005982700

C.i)
C.ii)
```

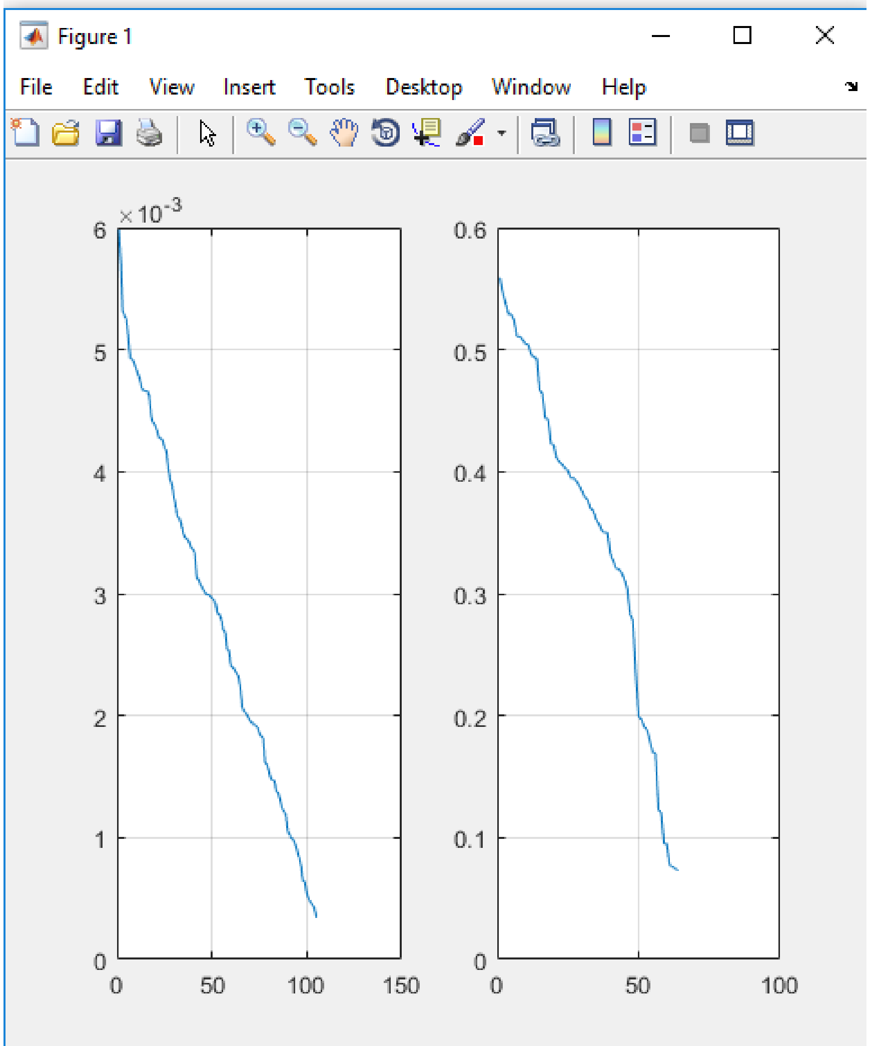

Figure 17: Results from Solution E

1.9 Question 2.i

We conclude then that the ISP should adopt the network that minimizes the max load. Around 2% off the max load is noticeable compared to the loss over the other methods, which are almost unrecognizable.