Modelação e Desempenho de Redes e Serviços

Second mini-project report

Bruno Gomes, Diogo Silva

Universidade de Aveiro

Modelação e Desempenho de Redes e Serviços

DEPARTAMENTO DE ELETRÓNICA
TELECOMUNICAÇÕES E INFORMÁTICA

Second mini-project report

Bruno Gomes, Diogo Silva
(103320) brunofgomes@ua.pt, (104341) diogobranco.as@ua.pt

23/10/2024

Contents

List of Figures

Chapter 1

Task 1

1.1 Exercise 1.a)

1.1.1 Code

```
load('InputDataProject2.mat')
nNodes= size(Nodes,1);
nLinks= size(Links,1);
nFlows= size(T,1);

% a)
% anycast services are provided by network nodes 3 and
    10
fprintf("1.a)\n");

anycastNodes = [3 10];
D = L/2e5;

Taux = zeros(nFlows,4);

% Computing up to k=1 shortest paths for all flows
    from 1 to nFlows:
k= 1;
sP= cell(1,nFlows);
nSP= zeros(1,nFlows);
for f=1:nFlows
    if T(f,1) == 1 || T(f,1) == 2  % unicast service
        [shortestPath, totalCost] = kShortestPath(D,T(
            f,2),T(f,3),k);
        sP{f}= shortestPath;
        nSP(f)= length(totalCost);
        Taux(f,:) = T(f,2:5);
    elseif T(f,1) == 3 % anycast service
        if ismember(T(f,2),anycastNodes)
            sP{f} = {T(f,2)};
            nSP(f) = 1;
            Taux(f,:) = T(f,2:5);
            Taux(f,2) = T(f,2);
        else %comparar totalCosts do shortest paths
            custo = inf;
            Taux(f,:) = T(f,2:5);
            for i=anycastNodes
                [shortestPath, totalCost] =
                    kShortestPath(D,T(f,2),i,k);
                if totalCost<custo
                    sP{f} = shortestPath;
                    nSP(f) = 1;
                    custo = totalCost;
                    Taux(f,2) = i;
                end
            end
        end
    end
end


rttDelays = zeros(nFlows,1);

for f = 1:nFlows
    path = sP{f}{1};
    for i = 1:length(path)-1
        rttDelays(f) = rttDelays(f) + 2 * D(path(i),
            path(i+1));
    end
end

RTT_unicast1 = rttDelays(T(:,1) == 1);
RTT_unicast2 = rttDelays(T(:,1) == 2);

RTT_anycast = rttDelays(T(:,1) == 3);

worstRTT_unicast1 = max(RTT_unicast1);
avgRTT_unicast1 = mean(RTT_unicast1);

worstRTT_unicast2 = max(RTT_unicast2);
avgRTT_unicast2 = mean(RTT_unicast2);

```

```
worstRTT_anycast = max(RTT_anycast);
avgRTT_anycast = mean(RTT_anycast);

fprintf("Service unicast 1:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    worstRTT_unicast1*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    avgRTT_unicast1*1000);

fprintf("Service unicast 2:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    worstRTT_unicast2*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    avgRTT_unicast2*1000);

fprintf("Service anycast:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    worstRTT_anycast*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    avgRTT_anycast*1000);
```

The code above, was used to calculate the worst round-trip delay and the average round trip delay of each of both the unicast services and the anycast service.

To begin the analysis of the code, we first load the network data matrices from the provided file, "InputDataProject2.mat." This file contains the necessary data for the analysis. After that, we we get the number of links, nodes and flows of the network, this will be useful to calculate the desired statistics from the network.

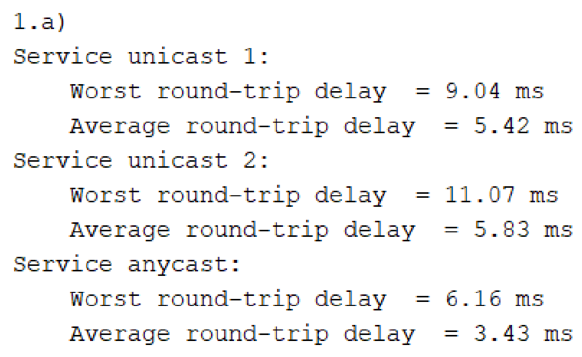
Then, we store the nodes 3 and 10 inside a vector that represents the chosen anycast nodes for this specific experiment. The matrix "D" that is calculated right after, represents the propagation delay on each direction of each link, the matrix "L" holds the values of the link lengths in kilometers, and the value "2e5" represents the value of the speed of light in the fibers.

Afterwards, we create a new "Taux" matrix that will be similar to the "T" matrix (the matrix "T" contains the flow characteristics of the network). However, in this specific case the "Taux" matrix will not have the first column that matrix "T" uses to identify which kind of service each flow supports, moreover, the destination of the anycast services will also change depending on the origin node, if the origin node belongs to an anycast provider, the destination of the flow will be that node itself, otherwise the shortest path will be calculated to determine the best destination anycast provider node. These details will be explained more in depth in the following section.

In the next part of the code, we calculate the shortest path of each flow based on their origin and destination nodes. We have "k = 1" because we only want the shortest path. Subsequently, we start calculating the shortest paths, each flow of the network is analyzed accordingly, we check if the flow supports unicast or anycast services, if the flow supports unicast services (if the first column of the matrix T is equal to 1 or 2) we store the shortest path found by the "kShortestPath.m" function, the number of shortest paths found will always be 1 because we defined "k=1" previously, and we create a new entry for the respective flow in the "Taux" matrix, without its type. If the flow supports the anycast service, we check if the origin node is part of the group of nodes responsible for providing the anycast service, if it is, the destination of the shortest path will be to the origin node itself, after storing the shortest path of the flow, we create a new entry in the "Taux" matrix which will have all the information related to the respective flow without its type, additionally we also update the destination node in that matrix to match the shortest path found previously for the specific anycast flow. There is also the case where the origin node will not be an anycast service provider, in this instance we need to find the best anycast provider for this node, for this we calculate 2 shortest paths for each different destination anycast node, then we compare the cost of both found paths, we keep the shortest one and we store the specific anycast destination node in the "Taux" matrix.

Lastly, we calculate the round-trip delay times. We iterate through the found shortest paths of each flow and we calculate the times between the pairs of nodes of each path, for this purpose we use the "D" matrix to get the delay times between different nodes of the network.

1.1.2 Results



```
1.a)
Service unicast 1:
    Worst round-trip delay   = 9.04 ms
    Average round-trip delay  = 5.42 ms
Service unicast 2:
    Worst round-trip delay   = 11.07 ms
    Average round-trip delay  = 5.83 ms
Service anycast:
    Worst round-trip delay   = 6.16 ms
    Average round-trip delay  = 3.43 ms
```

Figure 1.1: Round-trip times for all services with anycast nodes 3 and 10

1.2 Exercise 1.b)

1.2.1 Code

```
% b)
fprintf("1.b)\n");
% Compute the link loads using the first (shortest)
    path of each flow:
sol= ones(1,nFlows);
Loads= calculateLinkLoads(nNodes,Links,Taux,sP,sol);
% Determine the worst link load:
maxLoad= max(max(Loads(:,3:4)));

fprintf("Worst link load = %.2f Gbps\n",maxLoad);

for i = 1:nLinks
    fprintf('{%-2d - %-2d}         %8.2f %8.2f\n',
        Loads(i,1), Loads(i,2), Loads(i,3), Loads(i,4))
        ;
end
```

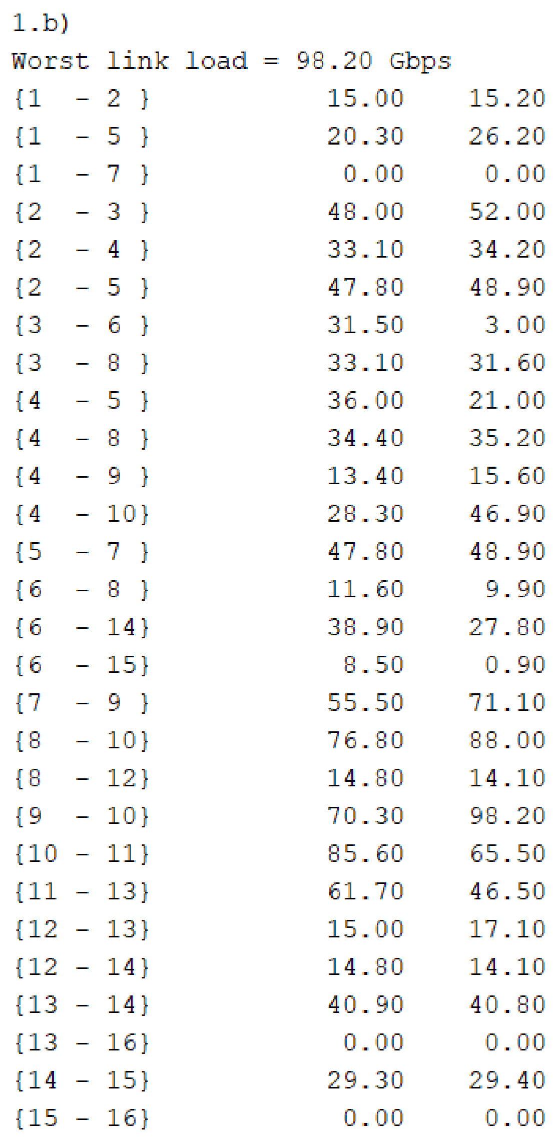
This code calculates the load on each network link as well as the maximum (worst) link load.

For this part, we used the function "calculateLinkLoads.m" that was provided to us in the practical classes, this function takes the number of nodes and links of the networks, takes the "Taux" matrix that we used previously, and a vector that holds the number of solutions (shortest paths) found for each flow.

After the function is finished running, we obtain a 28x4 matrix called "Loads". In this matrix, the first two columns represent the origin and destination nodes, while the last two columns contain the link loads: the load from the origin to the destination and the load from the destination to the origin.

To calculate the worst link load we filtered all the values present in columns 3 and 4 of the "Loads" matrix, and then we proceeded to find the biggest value present in them using the "max()" function provided by *MATLAB*.

1.2.2 Results



```
1.b)
Worst link load = 98.20 Gbps
{1  - 2 }          15.00    15.20
{1  - 5 }          20.30    26.20
{1  - 7 }           0.00     0.00
{2  - 3 }          48.00    52.00
{2  - 4 }          33.10    34.20
{2  - 5 }          47.80    48.90
{3  - 6 }          31.50     3.00
{3  - 8 }          33.10    31.60
{4  - 5 }          36.00    21.00
{4  - 8 }          34.40    35.20
{4  - 9 }          13.40    15.60
{4  - 10}          28.30    46.90
{5  - 7 }          47.80    48.90
{6  - 8 }          11.60     9.90
{6  - 14}          38.90    27.80
{6  - 15}           8.50     0.90
{7  - 9 }          55.50    71.10
{8  - 10}          76.80    88.00
{8  - 12}          14.80    14.10
{9  - 10}          70.30    98.20
{10 - 11}          85.60    65.50
{11 - 13}          61.70    46.50
{12 - 13}          15.00    17.10
{12 - 14}          14.80    14.10
{13 - 14}          40.90    40.80
{13 - 16}           0.00     0.00
{14 - 15}          29.30    29.40
{15 - 16}           0.00     0.00
```

Figure 1.2: Loads of all links and worst link load

1.3 Exercise 1.c)

1.3.1 Code

```
%c)
fprintf("1.c)\n");

% generate all possible combinations of two nodes
nodeCombinations = nchoosek(1:nNodes, 2);
nCombinations = size(nodeCombinations, 1);

% initialize variables to track the best combination
bestAnycastNodes = [];
minWorstLinkLoad = inf;

Taux = zeros(nFlows,4);

% computing up to k=1 shortest paths for all flows
    from 1 to nFlows:
k= 1;

for c = 1:nCombinations
    anycastNodes = nodeCombinations(c,:);
    sP= cell(1,nFlows);
    nSP= zeros(1,nFlows);
    for f=1:nFlows
        if T(f,1) == 1 || T(f,1) == 2  % unicast
            service
            [shortestPath, totalCost] = kShortestPath(
               D,T(f,2),T(f,3),k);
            sP{f}= shortestPath;
            nSP(f)= length(totalCost);
            Taux(f,:) = T(f,2:5);
        elseif T(f,1) == 3 % anycast service
            if ismember(T(f,2),anycastNodes)
                sP{f} = {T(f,2)};
                nSP(f) = 1;
                Taux(f,:) = T(f,2:5);
                Taux(f,2) = T(f,2);
            else %comparar totalCosts do shortest
               paths
                custo = inf;
                Taux(f,:) = T(f,2:5);
                for i=anycastNodes
                    [shortestPath, totalCost] =
                       kShortestPath(D,T(f,2),i,k);
                    if totalCost<custo
                        sP{f} = shortestPath;
                        nSP(f) = 1;
                        custo = totalCost;
                        Taux(f,2) = i;
                    end
                end
            end
        end
    end

    %compute link loads
    sol= ones(1,nFlows);
    Loads= calculateLinkLoads(nNodes,Links,Taux,sP,sol);
    worstLinkLoad= max(max(Loads(:,3:4)));

    %update the best anycast node combination
    if worstLinkLoad < minWorstLinkLoad
        minWorstLinkLoad = worstLinkLoad;
        bestAnycastNodes = anycastNodes;
    end

end


fprintf("Selected Anycast Nodes: %d %d\n", bestAnycastNodes(1),bestAnycastNodes(2));

% compute the round-trip delays for the best combination
sP= cell(1,nFlows);
nSP= zeros(1,nFlows);
for f=1:nFlows
    if T(f,1) == 1 || T(f,1) == 2  % unicast service
        [shortestPath, totalCost] = kShortestPath(D,T(f,2),T(f,3),k);
        sP{f}= shortestPath;
        nSP(f)= length(totalCost);
        Taux(f,:) = T(f,2:5);
    elseif T(f,1) == 3 % anycast service
        if ismember(T(f,2),bestAnycastNodes)
            sP{f} = {T(f,2)};
            nSP(f) = 1;
            Taux(f,:) = T(f,2:5);
            Taux(f,2) = T(f,2);
        else %comparar totalCosts do shortest paths
            custo = inf;
            Taux(f,:) = T(f,2:5);
            for i=bestAnycastNodes
                [shortestPath, totalCost] =
                   kShortestPath(D,T(f,2),i,k);
                if totalCost<custo
                    sP{f} = shortestPath;
                    nSP(f) = 1;
                    custo = totalCost;
                    Taux(f,2) = i;
                end
            end
        end
    end
end

rttDelays = zeros(nFlows,1);
% calculate rtt delays
for f = 1:nFlows
    path = sP{f}{1};
    for i = 1:length(path)-1
        rttDelays(f) = rttDelays(f) + 2 * D(path(i),
            path(i+1));
    end
end
%delay service by type
RTT_unicast1 = rttDelays(T(:,1) == 1);
RTT_unicast2 = rttDelays(T(:,1) == 2);

RTT_anycast = rttDelays(T(:,1) == 3);

worstRTT_unicast1 = max(RTT_unicast1);
avgRTT_unicast1 = mean(RTT_unicast1);

worstRTT_unicast2 = max(RTT_unicast2);
avgRTT_unicast2 = mean(RTT_unicast2);

worstRTT_anycast = max(RTT_anycast);
avgRTT_anycast = mean(RTT_anycast);

fprintf("Service unicast 1:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    worstRTT_unicast1*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    avgRTT_unicast1*1000);
```

```

fprintf("Service unicast 2:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    worstRTT_unicast2*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    avgRTT_unicast2*1000);

fprintf("Service anycast:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    worstRTT_anycast*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    avgRTT_anycast*1000);

fprintf("Worst link load = %.2f Gbps\n",
    minWorstLinkLoad);
```

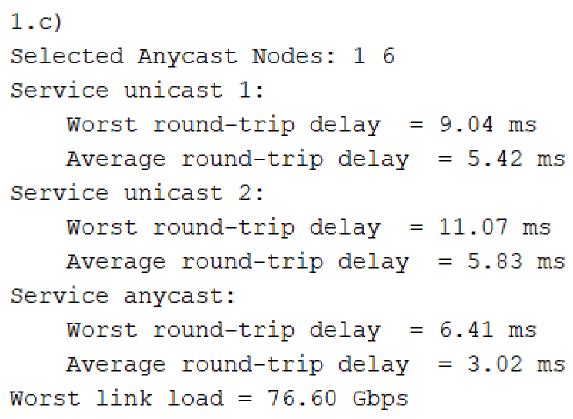
This code is very similar to the code developed in exercise **1.a.**, the only difference is that this code calculates link loads for all pairs of anycast nodes, with the main objective of finding out which pair of nodes generates the least maximum load in the network links.

First off, we start by generating all combinations of anycast nodes pairs, for this effect we use the function "nchoosek" which basically generates all combinations of a set of values. Additionally, we also initialize two auxiliary variables, the "bestAnycastNodes" vector will store the current best anycast nodes found by the algorithm, and the "minWorstLinkLoad" variable will be used to compare the subsequent loads computed by the algorithm for the different pairs of anycast nodes.

After the main algorithm computes the shortest path for a flow, the link loads for the network are then calculated, and the worst link load value is determined. This value is compared to the auxiliary variable "minWorstLinkLoad," which was previously initialized with an infinite value. If the computed worst link load is smaller than the value stored in the auxiliary variable, the auxiliary variable is updated to hold the new, lower value. And after several iterations through each pairs of anycast nodes we eventually get the optimal pair.

After getting the optimal pair of anycast nodes, we then run the same algorithm as in exercise **1.a.** and we calculate the desired times.

1.3.2 Results



```
1.c)
Selected Anycast Nodes: 1 6
Service unicast 1:
    Worst round-trip delay  = 9.04 ms
    Average round-trip delay  = 5.42 ms
Service unicast 2:
    Worst round-trip delay  = 11.07 ms
    Average round-trip delay  = 5.83 ms
Service anycast:
    Worst round-trip delay  = 6.41 ms
    Average round-trip delay  = 3.02 ms
Worst link load = 76.60 Gbps
```

Figure 1.3: Selected anycast nodes to minimize worst link load

1.4 Exercise 1.d)

1.4.1 Code

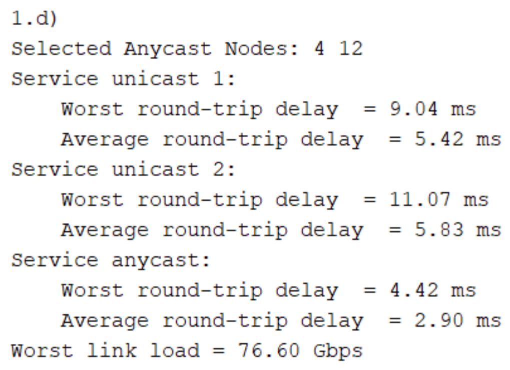
```
%d)
fprintf("1.d)\n");

minWorstRTT_anycast = inf;
bestAnycastNodes = [];

Taux = zeros(nFlows,4);

for c = 1:nCombinations
    anycastNodes = nodeCombinations(c,:);
    sP= cell(1,nFlows);
    nSP= zeros(1,nFlows);
    for f=1:nFlows
        if T(f,1) == 1 || T(f,1) == 2  % unicast service
            [shortestPath, totalCost] = kShortestPath(D,T(f,2),T(f,3),k);
            sP{f}= shortestPath;
            nSP(f)= length(totalCost);
            Taux(f,:) = T(f,2:5);
        elseif T(f,1) == 3 % anycast service
            if ismember(T(f,2),anycastNodes)
                sP{f} = {T(f,2)};
```

```matlab
                nSP(f) = 1;
                Taux(f,:) = T(f,2:5);
                Taux(f,2) = T(f,2);
            else %comparar totalCosts do shortest paths
                custo = inf;
                Taux(f,:) = T(f,2:5);
                for i=anycastNodes
                    [shortestPath, totalCost] = kShortestPath(D,T(f,2),i,k);
                    if totalCost<custo
                        sP{f} = shortestPath;
                        nSP(f) = 1;
                        custo = totalCost;
                        Taux(f,2) = i;
                    end
                end
            end
        end
    end

    rttDelays = zeros(nFlows,1);
    % calculate rtt delays
    for f = 1:nFlows
        path = sP{f}{1};
        for i = 1:length(path)-1
            rttDelays(f) = rttDelays(f) + 2 * D(path(i), path(i+1));
        end
    end

    RTT_unicast1 = rttDelays(T(:, 1) == 1);
    RTT_unicast2 = rttDelays(T(:, 1) == 2);
    RTT_anycast = rttDelays(T(:, 1) == 3);

    worstRTT_anycast = max(RTT_anycast);

    if worstRTT_anycast < minWorstRTT_anycast
        minWorstRTT_anycast = worstRTT_anycast;
        bestAnycastNodes = anycastNodes;
        bestRTT_unicast1 = RTT_unicast1;
        bestRTT_unicast2 = RTT_unicast2;
        bestRTT_anycast = RTT_anycast;

        %compute link loads
        sol= ones(1,nFlows);
        Loads= calculateLinkLoads(nNodes,Links,Taux,sP
            ,sol);
        maxLoad= max(max(Loads(:,3:4)));


    end

end

fprintf('Selected Anycast Nodes: %d %d\n',
    bestAnycastNodes(1), bestAnycastNodes(2));

fprintf("Service unicast 1:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",max(
    bestRTT_unicast1)*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",mean
    (bestRTT_unicast1)*1000);

fprintf("Service unicast 2:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",max(
    bestRTT_unicast2)*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",mean
    (bestRTT_unicast2)*1000);

fprintf("Service anycast:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",
    minWorstRTT_anycast*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",mean
    (bestRTT_anycast)*1000);

fprintf("Worst link load = %.2f Gbps\n",maxLoad);
```

The developed code for this exercise, is identical to the solution that was implemented for exercise **1.c.** with the main difference that in this specific case we are looking for the pair of anycast nodes that are able to minimize the worst round-trip delay of the anycast service.

In summary, after the algorithm calculates the shortest path for a given flow, we compute the round-trip delays for all the services and the worst link load of the network. The calculated worst round-trip delay for the anycast service is then compared to the auxiliary variable "minWorstRTT_anycast." If the calculated value is smaller, the auxiliary variable is updated to store this new value. After several iterations, we obtain the minimal worst round-trip delay, and worst link load for the selected anycast nodes.

1.4.2 Results



```
1.d)
Selected Anycast Nodes: 4 12
Service unicast 1:
    Worst round-trip delay  = 9.04 ms
    Average round-trip delay  = 5.42 ms
Service unicast 2:
    Worst round-trip delay  = 11.07 ms
    Average round-trip delay  = 5.83 ms
Service anycast:
    Worst round-trip delay  = 4.42 ms
    Average round-trip delay  = 2.90 ms
Worst link load = 76.60 Gbps
```

Figure 1.4: Selected anycast nodes to minimize worst round-trip delay of the anycast service

1.5 Exercise 1.e)

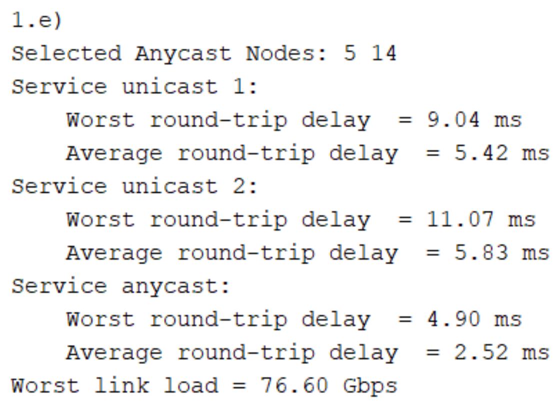
1.5.1 Code

```
%e)
fprintf("1.e)\n");

minAvgRTT_anycast = inf;
bestAnycastNodes = [];

Taux = zeros(nFlows,4);

for c = 1:nCombinations
    anycastNodes = nodeCombinations(c,:);
    sP= cell(1,nFlows);
    nSP= zeros(1,nFlows);
    for f=1:nFlows
        if T(f,1) == 1 || T(f,1) == 2  % unicast service
            [shortestPath, totalCost] = kShortestPath(D,T(f,2),T(f,3),k);
            sP{f}= shortestPath;
            nSP(f)= length(totalCost);
            Taux(f,:) = T(f,2:5);
        elseif T(f,1) == 3 % anycast service
            if ismember(T(f,2),anycastNodes)
                sP{f} = {T(f,2)};
```

```matlab
                nSP(f) = 1;
                Taux(f,:) = T(f,2:5);
                Taux(f,2) = T(f,2);
            else %comparar totalCosts do shortest paths
                custo = inf;
                Taux(f,:) = T(f,2:5);
                for i=anycastNodes
                    [shortestPath, totalCost] = kShortestPath(D,T(f,2),i,k);
                    if totalCost<custo
                        sP{f} = shortestPath;
                        nSP(f) = 1;
                        custo = totalCost;
                        Taux(f,2) = i;
                    end
                end
            end
        end
    end

    rttDelays = zeros(nFlows,1);
    % calculate rtt delays
    for f = 1:nFlows
        path = sP{f}{1};
        for i = 1:length(path)-1
            rttDelays(f) = rttDelays(f) + 2 * D(path(i), path(i+1));
        end
    end

    RTT_unicast1 = rttDelays(T(:, 1) == 1);
    RTT_unicast2 = rttDelays(T(:, 1) == 2);
    RTT_anycast = rttDelays(T(:, 1) == 3);

    avgRTT_anycast = mean(RTT_anycast);

    if avgRTT_anycast < minAvgRTT_anycast
        minAvgRTT_anycast = avgRTT_anycast;
        bestAnycastNodes = anycastNodes;
        bestRTT_unicast1 = RTT_unicast1;
        bestRTT_unicast2 = RTT_unicast2;
        bestRTT_anycast = RTT_anycast;

        %compute link loads
        sol= ones(1,nFlows);
        Loads= calculateLinkLoads(nNodes,Links,Taux,sP
            ,sol);
        maxLoad= max(max(Loads(:,3:4)));

    end

end

fprintf('Selected Anycast Nodes: %d %d\n',
    bestAnycastNodes(1), bestAnycastNodes(2));

fprintf("Service unicast 1:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",max(
    bestRTT_unicast1)*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",mean
    (bestRTT_unicast1)*1000);

fprintf("Service unicast 2:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",max(
    bestRTT_unicast2)*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",mean
    (bestRTT_unicast2)*1000);

fprintf("Service anycast:\n")
fprintf("\tWorst round-trip delay  = %.2f ms\n",max(
    bestRTT_anycast)*1000);
fprintf("\tAverage round-trip delay  = %.2f ms\n",
    minAvgRTT_anycast*1000);

fprintf("Worst link load = %.2f Gbps\n",maxLoad);
```

This code is essentially a modified version of the code used in exercise **1.d.**, with the key difference being that here we focus on finding the pair of anycast nodes that minimize the average round-trip delay of the anycast service. The only change made is in the variable being compared, following the calculation of the shortest paths for each flow. No additional explanations are required, as the rest of the code remains unchanged.

1.5.2 Results



```
1.e)
Selected Anycast Nodes: 5 14
Service unicast 1:
    Worst round-trip delay  = 9.04 ms
    Average round-trip delay  = 5.42 ms
Service unicast 2:
    Worst round-trip delay  = 11.07 ms
    Average round-trip delay  = 5.83 ms
Service anycast:
    Worst round-trip delay  = 4.90 ms
    Average round-trip delay  = 2.52 ms
Worst link load = 76.60 Gbps
```

Figure 1.5: Selected anycast nodes to minimize average round-trip delay of the anycast service

1.6 Exercise 1.f)

1.6.1 Conclusions

General Observations

1. Traffic Behavior of Unicast Services:

    - The round-trip delays of unicast services remain constant across all tasks. This is expected because the origin nodes of these services do not request services from anycast nodes, therefore anycast node selection is irrelevant for round-trip delays of unicast services.
    - The worst round-trip delay for unicast service 1 is always **9.04 ms**, and the average round-trip delay is always **5.42 ms**.
    - The worst round-trip delay for unicast service 2 is always **11.07 ms**, and the average round-trip delay is always **5.83 ms**.

2. Anycast Service Sensitivity:

    - The round-trip delays and link loads are directly influenced by the placement of anycast nodes. Optimization of anycast node placement impacts the performance metrics of the anycast service.

3. Link Load Behavior:

    - The worst link load in **Task 1.b** (fixed anycast nodes 3 and 10) is **98.20 Gbps**.

- When anycast nodes are optimized (in Task 1.c, 1.d, and 1.e), the worst link load reduces significantly to **76.60 Gbps**, showing that careful placement of anycast nodes helps alleviate network congestion.

Task-Specific Insights

- Task 1.a and 1.b (Baseline with fixed anycast nodes 3 and 10):
  - Worst link load: **98.20 Gbps**.
  - Anycast round-trip delays:
    - **Worst**: 6.16 ms
    - **Average**: 3.43 ms
  - This configuration results in the highest congestion (worst link load) and relatively poorer delay performance for the anycast service compared to optimized configurations.
- Task 1.c (Optimizing to minimize worst link load):
  - Selected anycast nodes: **1 and 6**.
  - The worst link load reduces to **76.60 Gbps**, a significant improvement over the baseline.
  - Anycast round-trip delays:
    - **Worst**: 6.41 ms (slightly worse that baseline)
    - **Average**: 3.02 ms (improved compared to baseline)
  - **Conclusion**: By selecting nodes **1 and 6**, the link loads are well-balanced, reducing network congestion, but at the cost of a marginal increase in the worst round-trip delay.
- Task 1.d (Optimizing to minimize worst round-trip delay for the anycast service):
  - Selected anycast nodes: **4 and 12**.
  - The worst round-trip delay for the anycast service improves significantly to **4.42 ms** (compared to 6.16 ms in the baseline).
  - The average round-trip delay also improves slightly to **2.90 ms**.
  - Worst link load: **76.60 Gbps**, consistent with the configuration in Task 1.c.
  - **Conclusion**: This configuration achieves the best **worst round-trip delay** for the anycast service while maintaining the improved link load distribution.
- Task 1.e (Optimizing to minimize average round-trip delay for the anycast service):

- Selected anycast nodes: **5 and 14**.
- The average round-trip delay for the anycast service improves significantly to **2.52 ms** (the best among all configurations).
- The worst round-trip delay for the anycast service is **4.90 ms**, which is close to the result in Task 1.d.
- Worst link load: **76.60 Gbps**, consistent with the configurations in Tasks 1.c and 1.d.
- **Conclusion**: This configuration strikes a balance by achieving the best **average round-trip delay** for the anycast service without increasing link load.

Key Comparisons and Conclusions

1. Impact on Worst Link Load:
   - Optimizing anycast node placement (Tasks 1.c, 1.d, and 1.e) reduces the worst link load from **98.20 Gbps** to **76.60 Gbps**, improving network performance and reducing congestion.
2. Impact on Anycast Service Round-Trip Delays:
   - **Task 1.d (nodes 4 and 12)** achieves the best **worst round-trip delay** of **4.42 ms**.
   - **Task 1.e (nodes 5 and 14)** achieves the best **average round-trip delay of 2.52 ms**.
3. Trade-offs:
   - Task 1.c focuses on link load minimization and achieves balanced performance.
   - Task 1.d and 1.e optimize different delay metrics for the anycast service while maintaining the same (lower) link load.
   - The choice of optimization depends on whether minimizing delay (worst or average) or reducing link congestion is prioritized.

Chapter 2

Task 2

2.1 Exercise 2.a)

2.1.1 Code ex2.a) main

```

%% 2.a)
load('InputDataProject2.mat')

v = 2e5; %speed of ligh on fiber km/s
k = 6; % k number of shortest paths
tL = 30; % time Limit
anycastNodes = [3, 10];

nNodes = size(Nodes, 1);
nFlows = size(T,1);

%delay converted to ms
D = L / v;
D = D * 1e3;

sP = cell(1,nFlows);
nSP = zeros(1,nFlows);

T_aux = zeros(nFlows,4);

for f = 1:nFlows
    if T(f,1) == 1 || T(f,1) == 2 % unicast
        [shortestPath, totalCost] = kShortestPath(D, T
            (f,2), T(f,3), k);
        sP{f} = shortestPath;
        nSP(f) = length(totalCost);
        T_aux(f,:) = T(f,2:5);
    elseif T(f,1) == 3 % anycast service
        if ismember(T(f,2),anycastNodes)
            sP{f} = {T(f,2)};
            nSP(f) = 1;
            T_aux(f,:) = T(f,2:5);
            T_aux(f,2) = T(f,2);
        else %comparar totalCosts do shortest paths
            custo = inf;
            T_aux(f,:) = T(f,2:5);
            for i=anycastNodes
                [shortestPath, totalCost] =
                   kShortestPath(D,T(f,2),i,1);
                if totalCost<custo
                    sP{f} = shortestPath;
                    nSP(f) = 1;
                    custo = totalCost;
                    T_aux(f,2) = i;
                end
            end
        end
    end
end

% Multi-Start Hill Climbing Algorithm with Greedy
   Randomized Start
fprintf('Running Multi-Start Hill Climbing... \n');
tic;
[bestSol, ~, noCycles, bestTime, bestCycle] =
   MultiStartHillClimbingRandomizedGredy(nNodes, Links
   , T_aux, sP, nSP, tL);
elapsedTime = toc;

Loads= calculateLinkLoads(nNodes,Links,T_aux,sP,
   bestSol);
% Determine the worst link load:
worstLinkLoad = max(max(Loads(:,3:4)));

% Round-Trip Delays for all flows
roundTripDelays = zeros(nFlows, 1);
for f = 1:nFlows
    path = sP{f}{bestSol(f)};
    for i = 1:length(path)-1
        roundTripDelays(f) = roundTripDelays(f) + 2 *
           D(path(i), path(i+1));
    end
end

% Delay by service type
RTT_unicast1 = roundTripDelays(T(:,1) == 1);
RTT_unicast2 = roundTripDelays(T(:,1) == 2);
RTT_anycast = roundTripDelays(T(:,1) == 3);

% Worst and average delay
worstRTT_unicast1 = max(RTT_unicast1);
avgRTT_unicast1 = mean(RTT_unicast1);

worstRTT_unicast2 = max(RTT_unicast2);
avgRTT_unicast2 = mean(RTT_unicast2);

worstRTT_anycast = max(RTT_anycast);
avgRTT_anycast = mean(RTT_anycast);


% Display results
fprintf('Task 2.a Results:\n');
fprintf('Unicast Service 1:\n');
fprintf('\tWorst Round-Trip Delay: %.2f ms\n',
    worstRTT_unicast1);
fprintf('\tAverage Round-Trip Delay: %.2f ms\n',
    avgRTT_unicast1);
fprintf('Unicast Service 2:\n');
fprintf('\tWorst Round-Trip Delay: %.2f ms\n',
    worstRTT_unicast2);
fprintf('\tAverage Round-Trip Delay: %.2f ms\n',
    avgRTT_unicast2);
fprintf('Anycast Service:\n');
fprintf('\tWorst Round-Trip Delay: %.2f ms\n',
    worstRTT_anycast);
fprintf('\tAverage Round-Trip Delay: %.2f ms\n',
    avgRTT_anycast);
fprintf('Worst Link Load: %.2f Gbps\n', worstLinkLoad)
    ;

fprintf('Performance Metrics:\n');
fprintf('\tTotal Cycles Run: %d\n', noCycles);
fprintf('\tTime for Best Solution: %.2f seconds\n',
    bestTime);
fprintf('\tCycles for Best Solution: %d\n', bestCycle)
    ;
```

```
fprintf('\tTotal Time Elapsed: %.2f seconds\n',
    elapsedTime);
```

Similarly, like in exercise 1.a) we begin by loading the network matrices from the "InputDataProject2.mat", calculating the propagation delay matrix converted to milliseconds, creating the T_aux matrix and selecting the anycast nodes 3 and 10. Following the same methodology as in exercise 1.a) we calculate the shortest paths for all flows with k = 6 number of shortest paths based on the service type and insert the destination nodes for anycast flows to the T_aux matrix so that we can use it later.

After obtaining the "sP", "nSP" and "T_aux" we run a Multi-Start Hill Climbing algorithm with initial Greedy Randomized Solutions with a limit duration of 30 seconds with the objective of minimizing the worst link load. We will discuss the algorithm in sectionss 2.1.2 and in 2.1.3.

After obtaining the best routing solution from the algorithm, we run the "calculateLinkLoads" function to find the worst link load.

For each flow, the selected path is used to calculate the round-trip delay and extract from the 3 services the worst round-trip delay and average round-trip delay.

At the end we display the following metrics:

- Worst and average round-trip delays for Unicast Service 1, Unicast Service 2 and Anycast Service in ms.
- Worst link load in the network in Gbps.
- Algorithm performance metrics, including total cycles run, time elapsed and the cycle at which the best solution was found, both in seconds.

2.1.2 Code ex2.a) Multi Start Hill Climbing Randomized Greedy

```

function [bestSol, bestObjective, noCycles, bestTime,
    bestCycle] = MultiStartHillClimbingRandomizedGredy(
    nNodes, Links, T, sP, nSP, timeLimit)
     t = tic; % Start the timer
     nFlows = size(T, 1); % Number of flows
     bestObjective = inf; % Initialize best objective
         to infinity
     noCycles = 0; % Initialize cycle counter
     aux = 0; % Auxiliary variable to calculate average
          objective
     bestTime = 0;
     bestCycle = 0;
     while toc(t) < timeLimit
```

```
        % Initialize solution using greedy randomized approach
        sol = zeros(1, nFlows);
        ordem = randperm(nFlows); % Randomized order of flows
        for f = ordem
            temp = inf; % Temporary best load
            % Greedy selection of the best path for the flow
            for p = 1:nSP(f)
                sol(f) = p;
                Loads = calculateLinkLoads(nNodes, Links, T, sP, sol);
                load = max(max(Loads(:, 3:4)));
                if load < temp
                    temp = load;
                    best_p = p; % Keep track of the best path
                end
            end
            sol(f) = best_p; % Assign the best path to the solution
        end

        % Calculate initial load for the greedy randomized solution
        Loads = calculateLinkLoads(nNodes, Links, T, sP, sol);
        load = max(max(Loads(:, 3:4)));

        % Refine solution using Hill Climbing
        [sol, load] = HillClimbing(nNodes, Links, T, sP, nSP, sol, load);

        % Update best solution and objective
        noCycles = noCycles + 1;
        aux = aux + load;
        if load < bestObjective
            bestSol = sol;
            bestObjective = load;
            bestTime = toc(t);
            bestCycle = noCycles;
        end
    end
end
```

This algorithm incorporates randomized greedy initialization for starting solutions and uses Hill Climbing to iteratively improve them with the intent of minimizing the worst link load in the network.

1. Initialization: The algorithm starts by tracking its runtime and initializing counters like the number of Cycles and the Cycles for the best solution.
2. Multi-Start Loop: The algorithm runs multiple iterations as long as the runtime is less than the specified "timeLimit"
3. Randomized Greedy Initialization: A random order "ordem" of traffic flows is generated using randperm. For each flow in the randomized order:
   - All possible paths "p" for the flow are evaluated.
   - The path that minimizes the worst link load "temp" is selected as the best path "best_p"
   - This best path is stored in the current solution "sol"

   This randomized greedy initialization ensures diverse starting points for the Hill Climbing phase, improving the chances of finding better solutions.
4. Hill Climbing: The HillClimbing function is called (will discuss in 2.1.3)
5. Evaluation Once the Hill Climbing phase is complete:
   (a) The number of completed cycles is incremented
   (b) If the current solution improves upon the best solution found so far: Update the best solution, best objective and record the time and cycle when this solution was achieved
6. Loop termination: The loop continues until the time limit is reached.

2.1.3 Code ex2.a) Hill Climbing

```

function [sol, load] = HillClimbing(nNodes, Links, T, sP, nSP, sol, load)
    nFlows = size(T,1);

    % If load is not provided, calculate it
    if nargin < 7 || isempty(load)
        Loads = calculateLinkLoads(nNodes, Links, T, sP, sol);
        load = max(max(Loads(:, 3:4)));
    end

    % Initialize local best variables
    bestLocalLoad = load;
    bestLocalSol = sol;

    % Hill Climbing Strategy
    improved = true;
    while improved
        improved = false; % Reset improvement flag
        % Test each flow
        for flow = 1:nFlows
            % Test each path of the flow
            for path = 1:nSP(flow)
                if path ~= sol(flow)
                    % Change the path for that flow
                    auxSol = sol;
                    auxSol(flow) = path;
                    % Calculate loads
                    Loads = calculateLinkLoads(nNodes,
                        Links, T, sP, auxSol);
                    auxLoad = max(max(Loads(:, 3:4)));

                    % Check if the current load is
                        better than the best local load
                    if auxLoad < bestLocalLoad
                        bestLocalLoad = auxLoad;
                        bestLocalSol = auxSol;
                        improved = true; % An
                            improvement was found
                    end
                end
            end
        end

        % Update global solution and load if improved
        if improved
            load = bestLocalLoad;
            sol = bestLocalSol;
        end
    end
end
```

The HillClimbing function refines a given solution by iteratively improving it. The goal is to minimize the worst link load in a network by adjusting the routing paths for each traffic flow.

1. Initialization: If the current link load is not provided, the algorithm calculates it using the calculateLinkLoads function.

The initial solution and load are stored as "bestLocalSol" and "bestLocalLoad"

2. Hill Climbing Process: The algorithm uses an iterative strategy to improve the current solution. The search for improvements continues until no better solution is found.

3. Exploring Neighboring Solutions: For each "flow":

   - Each candidate path for the flow is tested except the one already in the current solution "sol(flow)"
   - The path assignment for the flow is temporarily changed "auxSol", and the network's link load are recalculated using "calculateLinkLoads"

4. Evaluating Improvements:

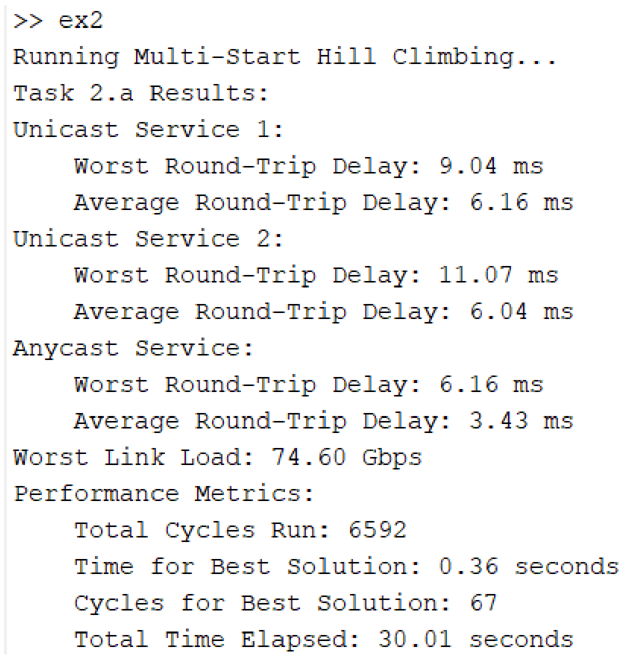
   - The worst link load "auxLoad" of the temporary solution is compared to "bestLocalLoad"
   - If the temporary solution improves the worst link load: Update the "bestLocalLoad" and "bestLocalSol" and set "improved" to true.

5. Updating the Solution: If an improvement is found during a full iteration over all flows and paths:

   - The "sol" and "load" are updated with the "bestLocalSol" and "bestLocalLoad"
   - The process repeats to further refine the solution

   If no improvement is found during an iteration the algorithm terminates

2.1.4 Results



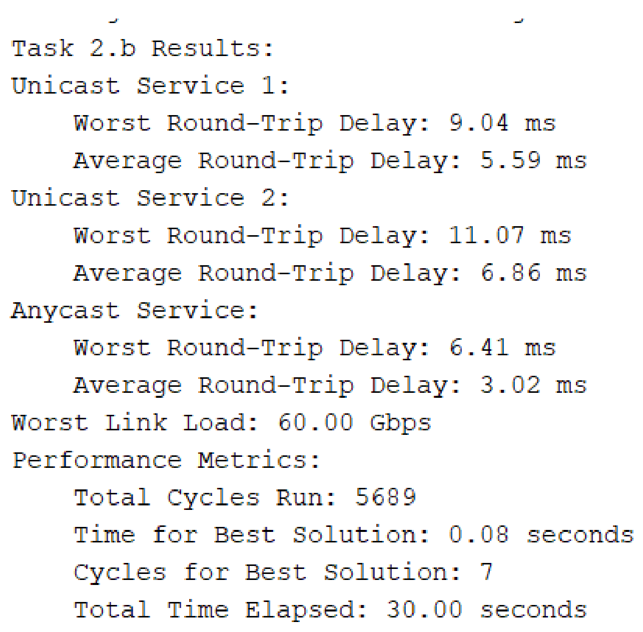
```
>> ex2
Running Multi-Start Hill Climbing...
Task 2.a Results:
Unicast Service 1:
    Worst Round-Trip Delay: 9.04 ms
    Average Round-Trip Delay: 6.16 ms
Unicast Service 2:
    Worst Round-Trip Delay: 11.07 ms
    Average Round-Trip Delay: 6.04 ms
Anycast Service:
    Worst Round-Trip Delay: 6.16 ms
    Average Round-Trip Delay: 3.43 ms
Worst Link Load: 74.60 Gbps
Performance Metrics:
    Total Cycles Run: 6592
    Time for Best Solution: 0.36 seconds
    Cycles for Best Solution: 67
    Total Time Elapsed: 30.01 seconds
```

Figure 2.1: Results of ex2.a)

2.2 Exercise 2.b)

Exercise 2.b)'s only difference from exercise 2.a) is the anycast Nodes which in this case are 1 and 6.

2.2.1 Results



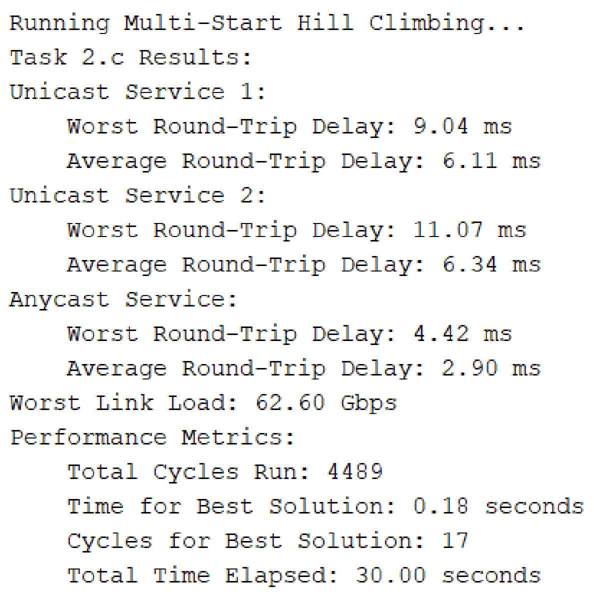
```
Task 2.b Results:
Unicast Service 1:
    Worst Round-Trip Delay: 9.04 ms
    Average Round-Trip Delay: 5.59 ms
Unicast Service 2:
    Worst Round-Trip Delay: 11.07 ms
    Average Round-Trip Delay: 6.86 ms
Anycast Service:
    Worst Round-Trip Delay: 6.41 ms
    Average Round-Trip Delay: 3.02 ms
Worst Link Load: 60.00 Gbps
Performance Metrics:
    Total Cycles Run: 5689
    Time for Best Solution: 0.08 seconds
    Cycles for Best Solution: 7
    Total Time Elapsed: 30.00 seconds
```

Figure 2.2: Results of ex2.b)

2.3 Exercise 2.c)

For this exercise the anycast nodes were 4 and 12

2.3.1 Results



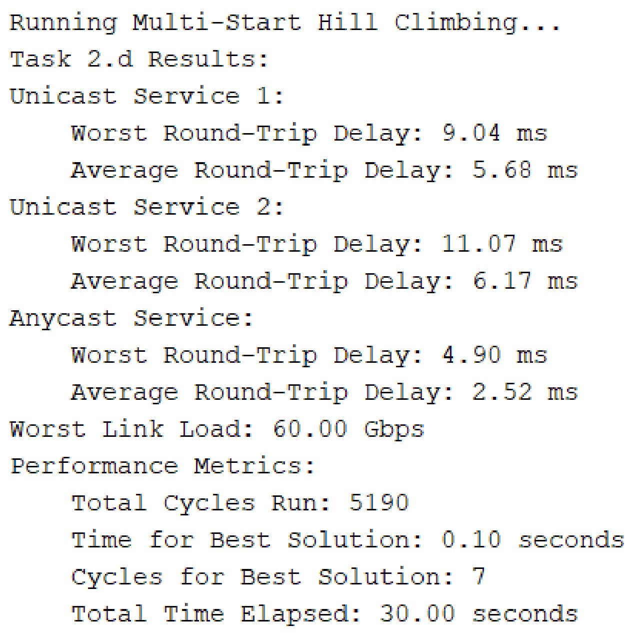
```
Running Multi-Start Hill Climbing...
Task 2.c Results:
Unicast Service 1:
    Worst Round-Trip Delay: 9.04 ms
    Average Round-Trip Delay: 6.11 ms
Unicast Service 2:
    Worst Round-Trip Delay: 11.07 ms
    Average Round-Trip Delay: 6.34 ms
Anycast Service:
    Worst Round-Trip Delay: 4.42 ms
    Average Round-Trip Delay: 2.90 ms
Worst Link Load: 62.60 Gbps
Performance Metrics:
    Total Cycles Run: 4489
    Time for Best Solution: 0.18 seconds
    Cycles for Best Solution: 17
    Total Time Elapsed: 30.00 seconds
```

Figure 2.3: Results of ex2.c)

2.4 Exercise 2.d)

In 2.d the anycast nodes were 5 and 14

2.4.1 Results



```
Running Multi-Start Hill Climbing...
Task 2.d Results:
Unicast Service 1:
    Worst Round-Trip Delay: 9.04 ms
    Average Round-Trip Delay: 5.68 ms
Unicast Service 2:
    Worst Round-Trip Delay: 11.07 ms
    Average Round-Trip Delay: 6.17 ms
Anycast Service:
    Worst Round-Trip Delay: 4.90 ms
    Average Round-Trip Delay: 2.52 ms
Worst Link Load: 60.00 Gbps
Performance Metrics:
    Total Cycles Run: 5190
    Time for Best Solution: 0.10 seconds
    Cycles for Best Solution: 7
    Total Time Elapsed: 30.00 seconds
```

Figure 2.4: Results of ex2.d)

2.5 Exercise 2.e)

1. Comparison of Worst Link Load: In Task 1, we computed the worst link load for different selections of anycast nodes:

3,10	:	98.2 Gbps
1,6	:	76.6 Gbps
4,12	:	76.6 Gbps
5,14	:	76.6 Gbps

The worst link load did not improve beyond 76.6 Gbps, however in Task 2, the worst link load was minimized further due to the Multi-Start Hill Climbing Algorithm:

3,10	:	74.6 Gbps
1,6	:	60.0 Gbps
4,12	:	62.6 Gbps
5,14	:	60.0 Gbps

This shows a significant improvement with the best improvement of 60.0 Gbps representing a 21.6% change compared to 76.6 Gbps in Task 1.

2. Comparison of Round-Trip Delays:
    - Service 1: In task 1, the worst delay was 9.04 ms, and remained unchanged across all experiments in Task 2. The average delay varied slightly, with the best result in Task 2.b (5.59 ms) and the worst in Task 2.a (6.16 ms)
    - Service 2: The worst delay for Service 2 was 11.07 ms in Task 1, and it remained unchanged for all experiments in Task 2. The average delay increased slightly in Task 2.b) (8.68 ms) compared to the best result in Task 2.a) (6.04 ms).
    - Service 3: The worst delay improved significantly with the optimal node selection (from 6.16 ms in Task 2.a to 4.42 ms in Task 2.c and 4.90 ms in Task 2.d). The average delay also improved for Task 2.c (2.90 ms) and Task 2.d (2.52 ms) compared to the baseline result of 3.43 ms in Task 2.a).

3. Final conclusions:
    - Task 2 achieved a significant improvment over Task 1, reduncing the worst link load to 60.0 Gbps in 2.b) and 2.d)
    - While RTT delays for Services 1 and 2 showed minor variations in Task 2, the delays for Service 3 (anycast) were optimized in Task 2.c) and Task 2.d).
    - Optimal anycast nodes in Task 2 (5,14) resulted in the best balance between link load reduction and delay performance

Chapter 3

Task 3

3.1 Exercise 3.a)

3.1.1 Code ex3.a) main

```
clear;
clc;
load('InputDataProject2.mat')

v = 2e5; % speed of light on fiber km/s
k = 12; % number of candidate paths
tL = 60; % time limit in seconds
anycastNodes = [3, 10];

nNodes = size(Nodes, 1);
nLinks = size(Links, 1);
nFlows = size(T, 1);

% delay converted to ms
D = L / v;
D = D * 1e3;

Taux = zeros(nFlows,4);

sP = cell(2, nFlows);
nSP = zeros(1, nFlows);

% compute candidate paths for unicast service s = 1, s
     = 2 and anycast
for f = 1:nFlows
    if T(f, 1) == 1 % unicast service s = 1
```

```
        [shortestPath, totalCost] = kShortestPath(D, T(f, 2), T(f, 3), k);
        sP{1,f} = shortestPath;
        sP{2,f} = {};
        nSP(f) = length(totalCost);
        Taux(f,:) = T(f,2:5);
    elseif T(f, 1) == 2 % unicast service s = 2
        [firstPaths, secondPaths, totalPairCosts] = kShortestPathPairs(D, T(f, 2), T(f, 3), k);
        %sP{f} = [firstPaths; secondPaths];
        sP{1,f} = firstPaths;
        sP{2,f} = secondPaths;
        nSP(f) = length(totalPairCosts);
        Taux(f,:) = T(f,2:5);
    elseif T(f, 1) == 3 % anycast service
        if ismember(T(f,2),anycastNodes)
            sP{1,f} = {T(f,2)};
            sP{2,f} = {};
            nSP(f) = 1;
            Taux(f,:) = T(f,2:5);
            Taux(f,2) = T(f,2);
        else
            custo = inf;
            Taux(f,:) = T(f,2:5);
            for i=anycastNodes
                [shortestPath, totalCost] = kShortestPath(D,T(f,2),i,1);
                if totalCost<custo
                    sP{1,f} = shortestPath;
                    sP{2,f} = {};
                    nSP(f) = 1;
                    custo = totalCost;
                    Taux(f,2) = i;
                end
            end
        end
    end
end

fprintf('Running Multi-Start Hill Climbing... \n');
tic;
[bestSol, bestObjective, noCycles, bestTime, bestCycle] = MultiStartHillClimbingRandomizedGredy3(nNodes, Links, Taux, sP, nSP, tL);
elapsedTime = toc;

```

```
% Compute link loads for the best solution
Loads = calculateLinkBand1to1(nNodes, Links, Taux, sP,
     bestSol);
worstLinkLoad = max(max(Loads(:, 3:4)));

% Compute round-trip delays for all flows
roundTripDelays = zeros(nFlows, 1);
for f = 1:nFlows
    path = sP{1,f}{bestSol(f)}; %compute RTT of
        working paths
    for i = 1:length(path)-1
        roundTripDelays(f) = roundTripDelays(f) + 2 *
            D(path(i), path(i+1));
    end
end

% Delay by service type
RTT_unicast1 = roundTripDelays(T(:, 1) == 1);
RTT_unicast2 = roundTripDelays(T(:, 1) == 2);
RTT_anycast = roundTripDelays(T(:, 1) == 3);

% Worst and average delay
worstRTT_unicast1 = max(RTT_unicast1);
avgRTT_unicast1 = mean(RTT_unicast1);

worstRTT_unicast2 = max(RTT_unicast2);
avgRTT_unicast2 = mean(RTT_unicast2);

worstRTT_anycast = max(RTT_anycast);
avgRTT_anycast = mean(RTT_anycast);


fprintf('Unicast Service 1:\n');
fprintf('\tWorst Round-Trip Delay: %.2f ms\n',
   worstRTT_unicast1);
fprintf('\tAverage Round-Trip Delay: %.2f ms\n',
   avgRTT_unicast1);
fprintf('Unicast Service 2:\n');
fprintf('\tWorst Round-Trip Delay: %.2f ms\n',
   worstRTT_unicast2);
fprintf('\tAverage Round-Trip Delay: %.2f ms\n',
   avgRTT_unicast2);
fprintf('Anycast Service:\n');
fprintf('\tWorst Round-Trip Delay: %.2f ms\n',
   worstRTT_anycast);
```

```
fprintf('\tAverage Round-Trip Delay: %.2f ms\n',
    avgRTT_anycast);
fprintf('Worst Link Load: %.2f Gbps\n', worstLinkLoad)
    ;

fprintf('Performance Metrics:\n');
fprintf('\tTotal Cycles Run: %d\n', noCycles);
fprintf('\tTime for Best Solution: %.2f seconds\n',
    bestTime);
fprintf('\tCycles for Best Solution: %d\n', bestCycle)
    ;
fprintf('\tTotal Time Elapsed: %.2f seconds\n',
    elapsedTime);
```

The code used for completing this task, is very similar to the code used for task 2, but here the unicast service 2 can have pairs of disjoint paths, the first path of the pair will be working path, while the second path will serve as the protection path. Each unicast service will have 12 shortest paths (since k="12"), however, service 1 will not have pairs of shortest paths. For the anycast service, the algorithm will only find the shortest path.

The process of finding the shortest paths for each flow had to altered, because the cell array that stores the shortest paths now has 2 rows, the first row will store the working paths, and the second row will store the protection paths. Basically, if the flow is part of the unicast service 1 or the anycast service, we store all the found shortest paths in the first row of the cell array, and since there these services do not support pairs of paths, in the second row we simply put an empty vector to avoid errors when calculating the link loads of the network.

The last important difference in this code, is that to calculate the link loads of the network we could not use the standard "calculateLinkLoads.m" function, since this function does not take into account that we can have pairs of paths, for this we used another function that was provided to us in one of our practical classes, which is the function "calculateLinkBand1to1.m", this function calculates the loads of the network properly, if the working paths stop working and the protection paths need to be activated.

3.1.2 Code ex3 Multi Start Hill Climbing Randomized Greedy

```
function [bestSol, bestObjective, noCycles, bestTime,
    bestCycle] = MultiStartHillClimbingRandomizedGredy3
    (nNodes, Links, T, sP, nSP, timeLimit)
    t = tic; % Start the timer
    nFlows = size(T, 1); % Number of flows
    bestObjective = inf; % Initialize best objective
        to infinity
    noCycles = 0; % Initialize cycle counter
    aux = 0; % Auxiliary variable to calculate average objective
    bestTime = 0;
    bestCycle = 0;
    while toc(t) < timeLimit
        % Initialize solution using greedy randomized approach
        sol = zeros(1, nFlows);
        ordem = randperm(nFlows); % Randomized order of flows
        for f = ordem
            temp = inf; % Temporary best load
            % Greedy selection of the best path for the flow
            for p = 1:nSP(f)
                sol(f) = p;
                Loads = calculateLinkBand1to1(nNodes, Links, T, sP, sol);
                load = max(max(Loads(:, 3:4)));
                if load < temp
                    temp = load;
                    best_p = p; % Keep track of the best path
                end
            end
            sol(f) = best_p; % Assign the best path to the solution
        end

        % Calculate initial load for the greedy randomized solution
        Loads = calculateLinkBand1to1(nNodes, Links, T, sP, sol);
        load = max(max(Loads(:, 3:4)));

        % Refine solution using Hill Climbing
        [sol, load] = HillClimbing3(nNodes, Links, T, sP, nSP, sol, load);

        % Update best solution and objective
        noCycles = noCycles + 1;
        aux = aux + load;
        if load < bestObjective
            bestSol = sol;
            bestObjective = load;
            bestTime = toc(t);
            bestCycle = noCycles;
        end
    end
end
```

This code is basically the same as in task 2, but we only change the way we calculate the link loads, for the reasons explained above.

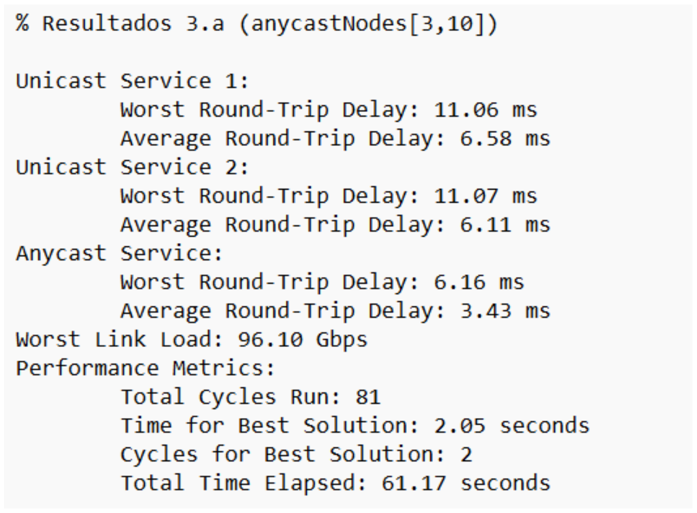
3.1.3 Code ex3 Hill Climbing

```
function [sol, load] = HillClimbing3(nNodes, Links, T,
    sP, nSP, sol, load)
    nFlows = size(T,1);

    % If load is not provided, calculate it
    if nargin < 7 || isempty(load)
        Loads = calculateLinkBand1to1(nNodes, Links, T
            , sP, sol);
        load = max(max(Loads(:, 3:4)));
    end

    % Initialize local best variables
    bestLocalLoad = load;
    bestLocalSol = sol;

    % Hill Climbing Strategy
    improved = true;
    while improved
        improved = false; % Reset improvement flag
        % Test each flow
        for flow = 1:nFlows
            % Test each path of the flow
            for path = 1:nSP(flow)
                if path ~= sol(flow)
                    % Change the path for that flow
                    auxSol = sol;
                    auxSol(flow) = path;
                    % Calculate loads
                    Loads = calculateLinkBand1to1(
                        nNodes, Links, T, sP, auxSol);
                    auxLoad = max(max(Loads(:, 3:4)));

                    % Check if the current load is
                        better than the best local load
                    if auxLoad < bestLocalLoad
```

```
                        bestLocalLoad = auxLoad;
                        bestLocalSol = auxSol;
                        improved = true; % An
                            improvement was found
                    end
                end
            end
        end

        % Update global solution and load if improved
        if improved
            load = bestLocalLoad;
            sol = bestLocalSol;
        end
    end
end
```

The Hill Climbing code is also similar to the code used for task 2, but here we also change the way we compute link loads.

3.1.4 Results

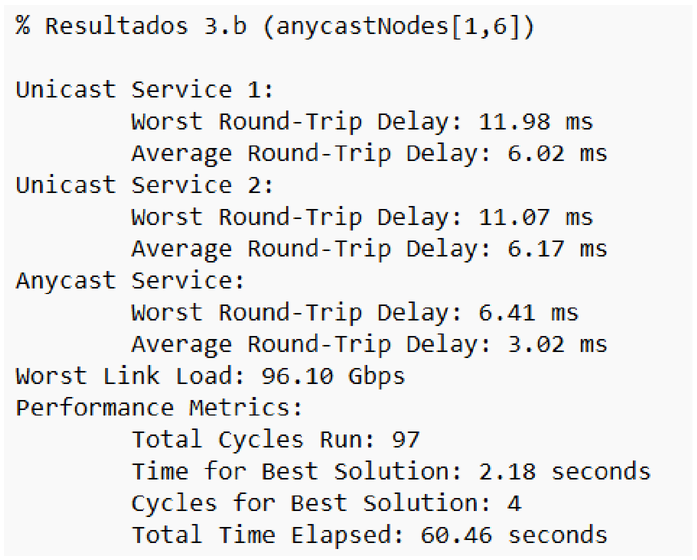


```
% Resultados 3.a (anycastNodes[3,10])

Unicast Service 1:
        Worst Round-Trip Delay: 11.06 ms
        Average Round-Trip Delay: 6.58 ms
Unicast Service 2:
        Worst Round-Trip Delay: 11.07 ms
        Average Round-Trip Delay: 6.11 ms
Anycast Service:
        Worst Round-Trip Delay: 6.16 ms
        Average Round-Trip Delay: 3.43 ms
Worst Link Load: 96.10 Gbps
Performance Metrics:
        Total Cycles Run: 81
        Time for Best Solution: 2.05 seconds
        Cycles for Best Solution: 2
        Total Time Elapsed: 61.17 seconds
```

Figure 3.1: Task 3.a results with anycast nodes 3 and 10

3.2 Exercise 3.b)

3.2.1 Results

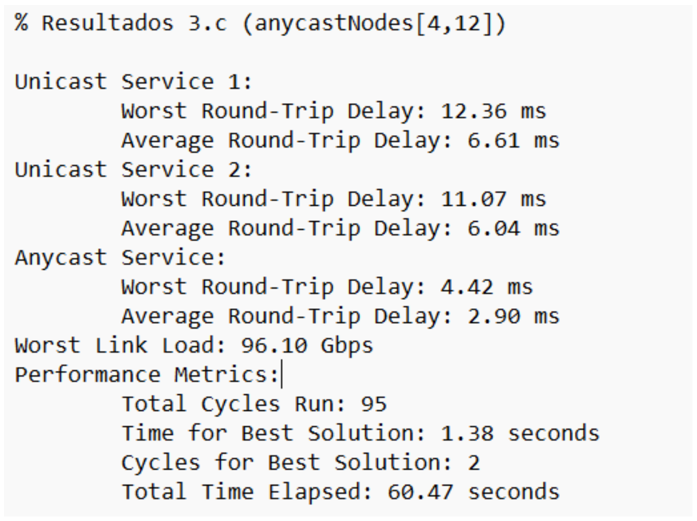


```
% Resultados 3.b (anycastNodes[1,6])

Unicast Service 1:
        Worst Round-Trip Delay: 11.98 ms
        Average Round-Trip Delay: 6.02 ms
Unicast Service 2:
        Worst Round-Trip Delay: 11.07 ms
        Average Round-Trip Delay: 6.17 ms
Anycast Service:
        Worst Round-Trip Delay: 6.41 ms
        Average Round-Trip Delay: 3.02 ms
Worst Link Load: 96.10 Gbps
Performance Metrics:
        Total Cycles Run: 97
        Time for Best Solution: 2.18 seconds
        Cycles for Best Solution: 4
        Total Time Elapsed: 60.46 seconds
```

Figure 3.2: Task 3.b results with anycast nodes 1 and 6

3.3 Exercise 3.c)

3.3.1 Results

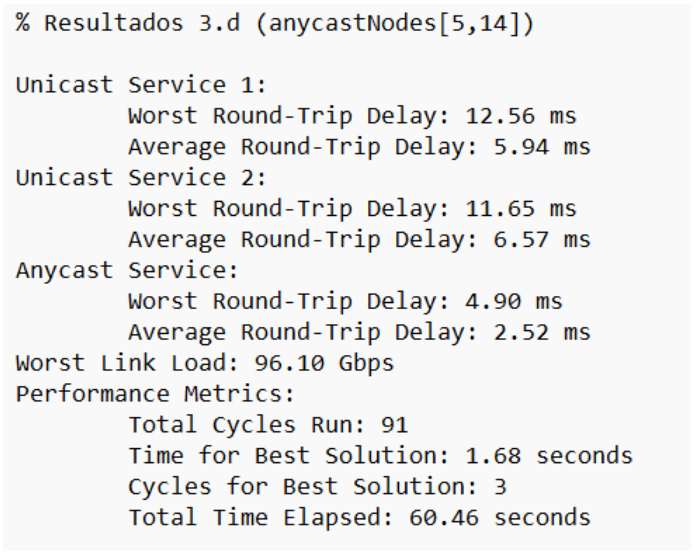


```
% Resultados 3.c (anycastNodes[4,12])

Unicast Service 1:
        Worst Round-Trip Delay: 12.36 ms
        Average Round-Trip Delay: 6.61 ms
Unicast Service 2:
        Worst Round-Trip Delay: 11.07 ms
        Average Round-Trip Delay: 6.04 ms
Anycast Service:
        Worst Round-Trip Delay: 4.42 ms
        Average Round-Trip Delay: 2.90 ms
Worst Link Load: 96.10 Gbps
Performance Metrics:
        Total Cycles Run: 95
        Time for Best Solution: 1.38 seconds
        Cycles for Best Solution: 2
        Total Time Elapsed: 60.47 seconds
```

Figure 3.3: Task 3.c results with anycast nodes 4 and 12

3.4 Exercise 3.d)

3.4.1 Results



```
% Resultados 3.d (anycastNodes[5,14])

Unicast Service 1:
        Worst Round-Trip Delay: 12.56 ms
        Average Round-Trip Delay: 5.94 ms
Unicast Service 2:
        Worst Round-Trip Delay: 11.65 ms
        Average Round-Trip Delay: 6.57 ms
Anycast Service:
        Worst Round-Trip Delay: 4.90 ms
        Average Round-Trip Delay: 2.52 ms
Worst Link Load: 96.10 Gbps
Performance Metrics:
        Total Cycles Run: 91
        Time for Best Solution: 1.68 seconds
        Cycles for Best Solution: 3
        Total Time Elapsed: 60.46 seconds
```

Figure 3.4: Task 3.d results with anycast nodes 5 and 14

3.5 Exercise 3.e)

3.5.1 Conclusions

1. Performance of the Algorithm

   - Across all experiments in **Task 3**, the algorithm achieved its best solution quickly, usually within the first few seconds:
     - **3.a**: Best solution achieved in **2.05 seconds** after **2 cycles**.
     - **3.b**: Best solution achieved in **2.18 seconds** after **4 cycles**.
     - **3.c**: Best solution achieved in **1.38 seconds** after **2 cycles**
     - **3.d**: Best solution achieved in **1.68 seconds** after **3 cycles**.
   - Compared to **Task 2**, the algorithm in **Task 3** required fewer cycles to reach the optimal solution. This is because, in **Task 3**, the algorithm performs fewer cycles within the 60-second time frame. Specifically, the multi-start algorithm in this task has more potential paths to explore, 12 possible shortest paths for unicast services, which is double the number of paths in **Task 2**. Additionally, in unicast service 2, the algorithm must consider path pairs, further increasing the complexity of the search for the best network solution, and therefore decreasing the number of total cycles that the algorithm can execute.

- **Time Elapsed**: The total time elapsed was consistently around 60 seconds in **Task 3**, indicating that the stopping time was adhered to.

2. Impact on Worst Link Load

    - The **worst link load** in **Task 3** was consistently **96.10 Gbps** across all experiments, significantly higher that in **Task 2**:
        - This increase is due to the additional load introduced by the **1:1 protection mechanism** in **Task 3**, which reserves resources for backup paths in unicast service 2.
        - In **Task 2**, the worst link load ranged from **60.00 Gbps** to **74.60 Gbps**, showing that without protection, the network was better optimized for load.

3. Anycast Performance Comparison

    - The **anycast service round-trip delays** remain unchanged between **Task 2** and **Task 3** across all experiments:
        - For instance:
            - **Task 2.d** (Anycast Nodes [5,14]):
                - Worst RTT: **4.90 ms**
                - Average RTT: **2.52 ms**
            - **Task 3.d** (Anycast Nodes [5,14])
                - Worst RTT: **4.90 ms**
                - Average RTT: **2.52 ms**
        - Similarly, for all other experiments, the anycast delays also remain constant.

    This suggests that the **1:1 protection mechanism for unicast service 2** in **Task 3** has no **direct impact on the anycast service performance**. The routing paths for the anycast service likely remain unaffected due to their unique routing strategy and lack of reliance on the protection mechanism.

4. Unicast Performance Comparison

    - For **unicast service 1** and **unicast service 2**, the worst and average round-trip delays increased slightly in **Task 3** compared to **Task 2**, due to the inclusion of the protection mechanism:
        - **Unicast Service 1**:
            - Worst RTTs increased slightly (11.06 ms in 3.a vs 9.04 ms in 2.a).
            - Average RTTs show similar increases (6.58 ms in 3.a vs 6.16 ms in 2.a).
        - **Unicast Service 2**:

        * Worst RTT remained **11.07 ms** across most experiments in both tasks, indicating that the protection mechanism's working paths may not significantly affect this metric.
        * Average RTTs show slight increases (6.11 ms in 3.a vs 6.04 ms in 2.a).

5. Impact of Anycast Node Selection
    - The selection of **anycast nodes** affects the performance metrics, particularly the worst round-trip delay and average round-trip delay for the anycast service:
        - **Task 3.d (Anycast Nodes 5,14)** achieved the best anycast RTT performance:
            * Worst RTT: **4.90 ms**
            * Average RTT: **2.52 ms**
        - **Task 3.c (Anycast Nodes 4,12)** was also efficient:
            * Worst RTT: **4.42 ms**
            * Average RTT: **2.90 ms**
        - These results align with **Task 2**, where the selection of anycast nodes 5 and 14 or 4 and 12 consistently resulted in better RTTs compared to nodes 3 and 10.

6. Overall Conclusions
    - Anycast Service Independence:
        - The **anycast service delays** (both worst and average) remain constant across all experiments in **Task 2** and **Task 3**, showing that the protection mechanism for unicast service 2 does not affect the anycast routing paths or their perfomance.
    - Unicast Service Impact:
        - The introduction of the **1:1 protection mechanism** in **Task 3** results in increased delays (both worst and average) for unicast services 1 and 2, as well as higher worst link load (96.10 Gbps).
    - Network Load Trade-Off:
        - Task 3 ensures fault tolerance for unicast service 2 at the expense of higher network load and slightly worse delay performance for unicast services.
    - Algorithm Efficiency:
        - The Multi-Start Hill Climbing algorithm performed more slowly in Task 3 when compared to Task 2, since it had to analyze more candidate paths.
    - Node Selection Consistency:
        - The results confirm that certain node pairs, consistently yield better anycast performance across both tasks, though the delays themselves remain constant.

Author's Contribution

Each member contributed equally to the development of this project, therefore, we decided to allocate 50% to each.