Universidade de Aveiro

Modelação e Desempenho de Redes e Serviços

Mini-Project nº2

Pedro Carneiro (73775)

Inês Águia (73882)

December 17th 2024

Index

1. Introduction

In alignment with the requirements of the mini-project for the course "Modeling and Performance of Networks and Services", this report presents a detailed analysis of the tasks performed, supported by key MATLAB code snippets. It demonstrates the rationale behind each solution and summarizes the conclusions deduced from the results of each exercise.

The report is divided into three chapters, one for each task and within each task into smaller sections for each exercise.

2. Task 1

2.1 Exercise: 1a

Required: Compute the worst round-trip delay and the average round trip delay of each of the 3 services (presenting all values in milliseconds) if the anycast nodes of the anycast service ($S = 3$) are network nodes 3 and 10.

Listing 2.1: MATLAB code for Data Load and vars initialization

```
load('InputDataProject2.mat') % ---> Load Data

nNodes= size(Nodes,1);
nFlows= size(T,1);
v = 2e5; % ---> speed of light on fibers
D = L/v; % ---> propagation delay on each direction of each link
anycastNodes = [3, 10]; %
roundTripDelays = zeros(1, nFlows);
Taux = zeros(1,4);
```

To analyze the propagation delays and network performance, the following steps were initiated like we can see on the Listing 2.1:

1. **Data Import**: the network topology data, including node position (Nodes), traffic flows (T), and link lengths (L) was loaded from the *InputDataProject2.mat*

2. **Network Parameters**:

   - The number of nodes and flows in the network were determined, with *nNodes* representing total number of nodes and *nFlows* representing the total traffic flows.
   - The propagation speed of signals in fiber-optic links was set using the Equation 2.1.

$$v = 2 \times 10^5 \text{ Km/s} \tag{2.1}$$

3. **Propagation Delay Calculation**: The *one-way* propagation delay for each link was calculated using Equation 2.2, where L represents the length of the link.

$$D = \frac{L}{v} \tag{2.2}$$

4. **Anycast Nodes**: Nodes 3 and 10 were designated as **anycast nodes**, allowing flows to be routed to the nearest of these nodes.

5. **Initialization for Delay Analysis**:

   - **roundTripDelays** initialized to store the round-trip propagation delays for each flow.
   - **Taux** is a temporary array to store the traffic for each flow, without the service column to be used in intermediate calculations.

Listing 2.2: MATLAB code for Path Selection and Delay Calculation

```
k = 1;
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); % ---> in this case 1
        Taux(f,:) = T(f,2:5);
        roundTripDelays(f) = 2 * totalCost * 1000; % ---> ms convertion
    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); % ---> Origin Node
        else
            Taux(f,:) = T(f,2:5);
            minCost = inf;
            for acNode = anycastNodes
                [shortestPath, totalCost] = kShortestPath(D, T(f,2), acNode, k);
                if totalCost < minCost % ---> Calulate the nearest node [3 or 10]
                    minCost = totalCost;
                    sP{f} = shortestPath;
                    nSP(f) = 1; % ---> in this case 1
                    Taux(f,2) = acNode; % ---> Origin node [3 10]
                end
            end
            roundTripDelays(f) = 2 * minCost * 1000; % ---> ms convertion
        end
    end
```

```
30 end
```

The network routing process was implemented to handle **unicast** and **anycast** services, calculating the shortest paths and corresponding delays for all traffic flows. The methodology for each type of service presented on Listing 2.2 is detailed below:

1. **Unicast Service** (Service 1 and 2): For unicast services, the shortest path between the source node and the destination node was determined using the **kShortestPath** function.
   - **Shortest Path Computation**:
     - The function was called with k=1, ensuring only the single shortest path was considered.
     - The result includes the shortest path (*shortestPath*) and its associated total cost (*totalCost*), which represents the one-way delay.
   - **Round-trip Calculation**:
     - Round-trip delay was calculated using the Equation 2.3

$$roundTripDelays(f) = 2 \times totalCost \times 1000 \tag{2.3}$$

     - This equation accounts for the two-way delay (round trip) and converts the result into milliseconds.
2. **Anycast Service** (Service 3): For anycast services, additional logic was implemented to handle multiple potential destination nodes (anycast nodes).
   - **Direct Anycast Handling**: If the source node was already one of the designated anycast nodes, it was used as the destination. In this case, no path calculation was needed, and the delay is zero,
   - **Shortest Path to Anycast Nodes**:
     - If the source node was not an anycast node, the shortest path to each anycast node was obtained using the **kShortestPath** function.
     - The anycast node with the smallest one-way delay (*totalCost*) was chosen as the destination.
   - **Round-trip Calculation**: The round-trip delay was calculated using Equation 2.3, similar to unicast services

With this implementation we got the results presented on the Table 2.1.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.04	5.42
2	11.07	5.83
3	6.16	3.43

Table 2.1: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [3 10]

2.2 Exercise: 1b

Required: Determine the link loads of all links and the worst link load of the network of the previous solution.

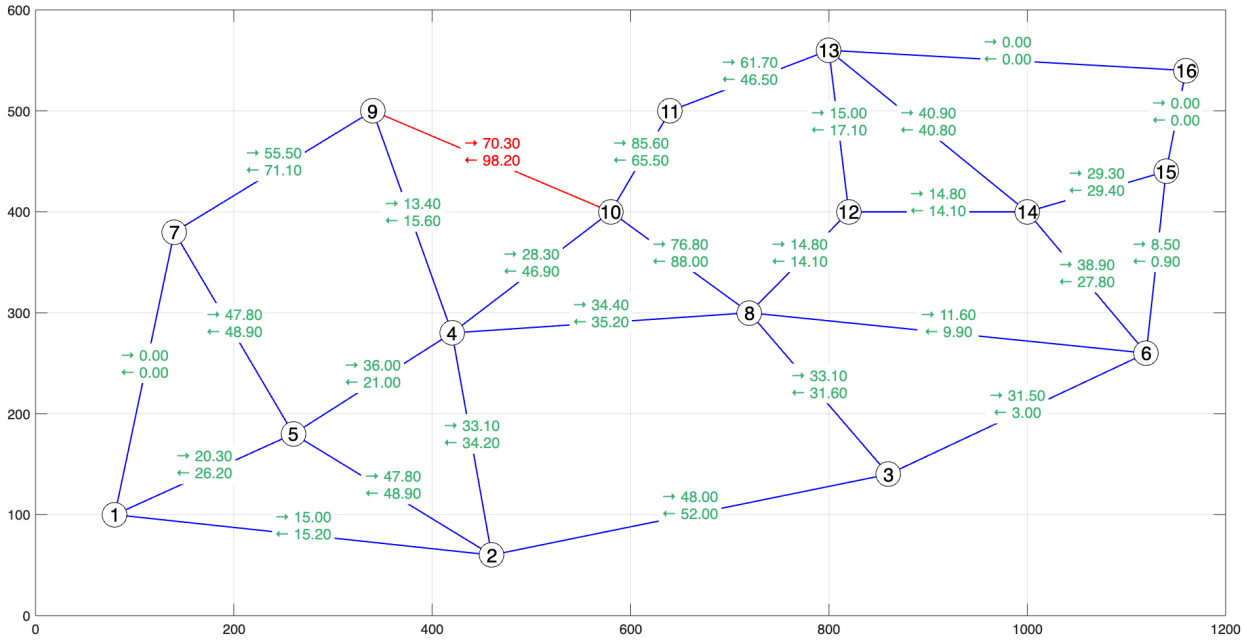

Figure 2.1: Visual representation of the Link Loads from the solution 1.a) where → represents source to destination and the ← destination to source

Listing 2.3: MATLAB code to determine the link loads of all links in the network from the previous solution

```
1    sol= ones(1,nFlows); % ---> Initialize vector all ones
2    Loads= calculateLinkLoads(nNodes,Links,Taux,sP,sol);
3
4    maxLoad= max(max(Loads(:,3:4))); % ---> Determine the worst link load:
```

To determine the link loads of all links and the worst link load in the network based on the previous solution, we used the code provided in Listing 2.2, which performs the following steps:

1. **Compute Link loads**:
    - Using the shortest paths (sP) and traffic flows (T) obtained from the previous solution, the function **calculateLinkLoads** was usse to calculate the load on each link.
    - This function returns a matrix, *Loads*, containing detailed information about the load on each link, including the traffic in both directions.
2. **Identify Worst Link Load**:
    - The worst link load, representing the highest traffic load across all network links, was identified from the Loads matrix, previously generated using the function **calculateLinkLoads**.
    - Specifically, the maximum value was extracted from columns 3 and 4 of the matrix, which contain the traffic loads in the two directions of each link.

This code accurately calculates the network's link loads, Table 2.2. By identifying **Worst Link-Load** = 98.20 Gbps, we can conclude that the network has a link load close to the maximum capacity which indicates a potencial bottleneck, as the link is near its full capacity.

Nodes	Link Load (Gbps)	
	Src → Dst	Dst → Src
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

Table 2.2: Table showing all the link loads for the solution in 1.a, in red color we got the WORST LINK LOAD and orange the second worst link load.

2.3 Exercise: 1c

Required: Consider that you can freely select the anycast nodes of the anycast service. Try all possible combinations of 2 nodes and select the one that minimizes the worst link load. Indicate the two selected nodes, the worst round-trip delay and the average round trip delay of each service.

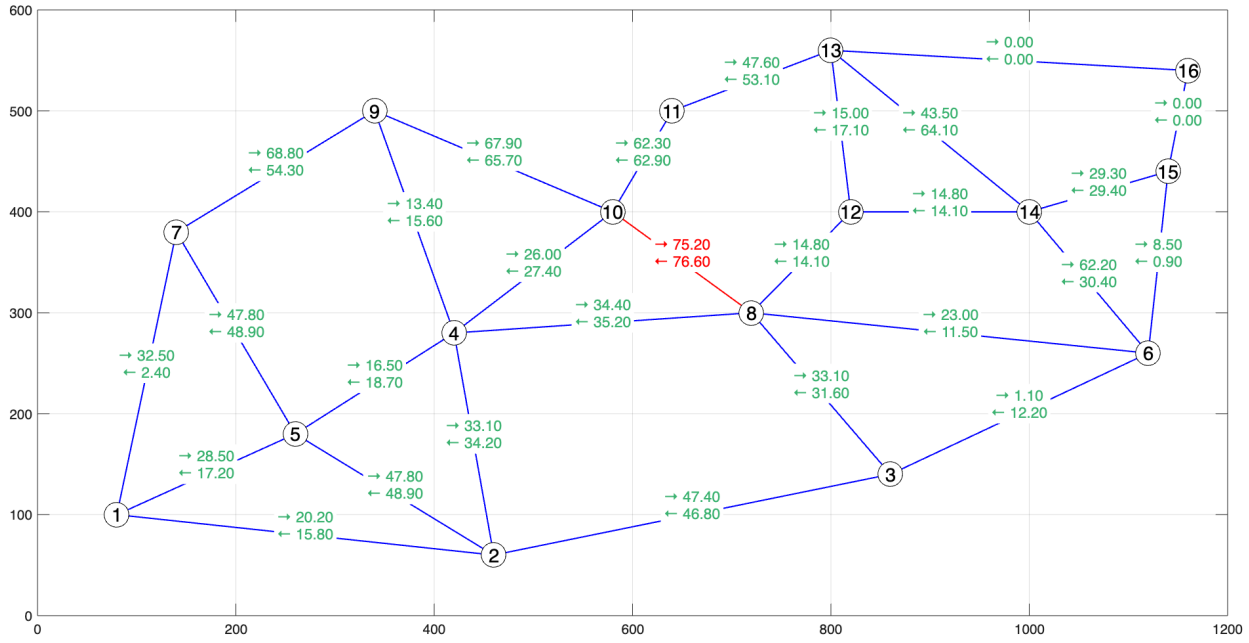

Figure 2.2: Visual representation of the Link Loads, using the nodes [1 6] as anycast nodes, where → represents source to destination and the ← destination to source

Listing 2.4: MATLAB code of the Optimization of Anycast Nodes and Worst Link Load

```
for n1 = 1:nNodes-1
    for n2 = n1+1:nNodes
        anycastNodes = [n1, n2];
        Taux = zeros(nFlows, 4);
        roundTripDelays = zeros(1, nFlows);
        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), 1);
                sP{f} = shortestPath;
                nSP(f) = length(totalCost);
                Taux(f,:) = T(f,2:5);
                roundTripDelays(f) = 2 * totalCost * 1000;
            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
                    Taux(f,:) = T(f,2:5);
```

```
                    minCost = inf;
                    for acNode = anycastNodes
                        [shortestPath, totalCost] = kShortestPath(D, T(f,2),
                            acNode, 1);
                        if totalCost < minCost
                            minCost = totalCost;
                            sP{f} = shortestPath;
                            nSP(f) = 1;
                            Taux(f,2) = acNode;
                        end
                    end
                    roundTripDelays(f) = 2 * minCost * 1000;
                end
            end
        end

        % Compute link loads
        sol = ones(1, nFlows);
        Loads = calculateLinkLoads(nNodes, Links, Taux, sP, sol);

        % Find worst link load
        maxLoad= max(max(Loads(:,3:4)));

        % Update best solution if current solution is better
        if maxLoad < bestWll
            bestWll = maxLoad;
            bestAnycastNodes = anycastNodes;
            bestRoundTripDelays = roundTripDelays;
        end
    end
end
```

The Listing 2.4 evaluates all possible pairs of nodes as potential anycast nodes to minimize the worst link-load in the network. This process systematically examines all pairs of nodes as anycast nodes candidates, as detailed in Listing 2.5.

Listing 2.5: MATLAB code to evaluate pairs of nodes

```
    for n1 = 1:nNodes-1
        for n2 = n1+1:nNodes
            anycastNodes = [n1, n2];
```

The loop in Listing 2.5 generates all pairs of nodes as potential anycast nodes, storing the current pair of nodes being evaluated, *anycastNodes*.

For each pair of nodes, the same procedure described in Exercise 1.a is applied:

1. Shortest paths and round-trip delays are calculated for all flows,

2. The *Loads* Matrix is calculated to determine the link loads for the current pair,

3. The maximum link load (*maxLoad*) is identified.

If the current maximum link load is lower than the previously stored best value (*bestWll*), the solution is updated to reflect the new optimal configuration. The process is shown in Listing 2.6.

Listing 2.6: MATLAB code to store the best pair of nodes that minimize the worst link load

```
% Update best solution if current solution is better
if maxLoad < bestWll
    bestWll = maxLoad;
    bestAnycastNodes = anycastNodes;
    bestRoundTripDelays = roundTripDelays;
end
```

Applying this solution the `best pair of anycast nodes is [1, 6]`, which minimizes the **worst link load in the network** to 76.60 Gbps, significantly improving network performance and reducing potential congestion.

The evaluation also provided insights into the round-trip delays for the services in the network. These metrics, including the worst and average round-trip delays for each service, are summarized in Table 2.3:

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.04	5.42
2	11.07	5.83
3	6.41	3.02

Table 2.3: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [1 6]

2.4 Exercise: 1d

Required: Again, consider that you can freely select the anycast nodes of the anycast service. Try all possible combinations of 2 nodes and select the one that minimizes the worst round-trip delay of the anycast service. Indicate the two selected nodes, the worst round-trip delay and the average round trip delay of each service.

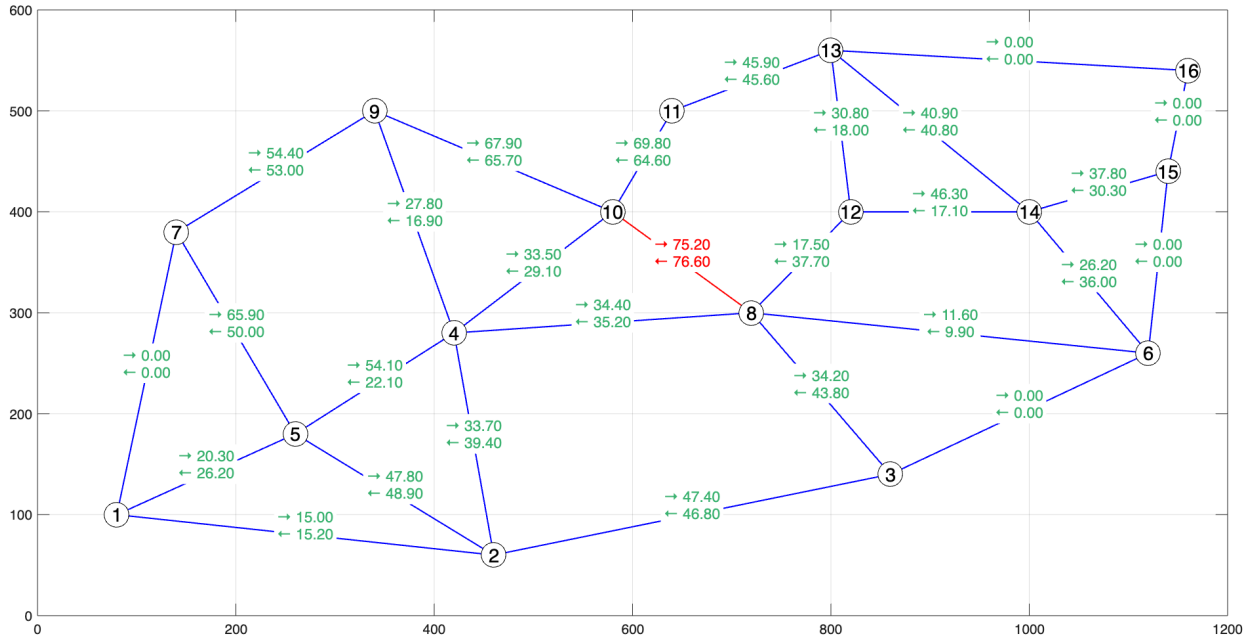

Figure 2.3: Visual representation of the Link Loads, using the nodes [4 12] as anycast nodes, where → represents sorce to destination and the ← destination to source

Listing 2.7: MATLAB code of the Optimization of Anycast Nodes and worst round-trip delay of the anycast service

```
for n1 = 1:nNodes-1
    for n2 = n1+1:nNodes
        anycastNodes = [n1, n2];
        Taux = zeros(nFlows, 4);
        roundTripDelays = zeros(1, nFlows);
        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), 1);
                sP{f} = shortestPath;
                nSP(f) = length(totalCost);
                Taux(f,:) = T(f,2:5);
                roundTripDelays(f) = 2 * totalCost * 1000;
            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
```

```
                    Taux(f,:) = T(f,2:5);
                    minCost = inf;
                    for acNode = anycastNodes
                        [shortestPath, totalCost] = kShortestPath(D, T(f,2),
                            acNode, 1);
                        if totalCost < minCost
                            minCost = totalCost;
                            sP{f} = shortestPath;
                            nSP(f) = 1;
                            Taux(f,2) = acNode;
                        end
                    end
                    roundTripDelays(f) = 2 * minCost * 1000;
                end
            end
        end

        % Compute link loads
        sol = ones(1, nFlows);
        Loads = calculateLinkLoads(nNodes, Links, Taux, sP, sol);

        % Find worst link load
        maxLoad= max(max(Loads(:,3:4)));

        % Find worst anycast service round-trip delay
        anycastDelays = roundTripDelays(T(:,1) == 3);
        worstAnycastDelay = max(anycastDelays);

        % Update best solution if current solution has lower worst anycast delay
        if worstAnycastDelay < bestWorstAnycastDelay
            bestWorstAnycastDelay = worstAnycastDelay;
            bestAnycastNodes = anycastNodes;
            bestRoundTripDelays = roundTripDelays;
            bestMaxLoad = maxLoad;
        end
    end
end
```

The code displayed in Listing 2.7 is identical to the one used in the previous exercises. It iteratively evaluates all pairs of nodes as potential anycast nodes to optimize network performance. However, the key difference in this exercise is that the objective has shifted: instead of minimizing the worst link-load, the focus is now on minimizing the **worst round-trip delay for the anycast service** (Service 3).

1. **Evaluation of All Node Pairs** (*Same as 1c*),

2. **Link Load Calculation** (*Same as 1c*),

3. **Focus on Anycast Service Delays** (New Objective):

   - Introducing a new variable, anycastDelays, which extracts the round-trip delays for Service 3 (anycast service) from the roundTripDelays array and the worst round-trip delay is identified as the maximum if the anycastDelays
   - Instead of choosing the anycast node pair based on minimizing the worst link-load, the decision now depends on the worst round-trip delay for the anycast service, as we can see on Listing 2.8.

Listing 2.8: MATLAB code to store the best pair of nodes that minimize the worst round-trip delay of the anycast service

```
    if worstAnycastDelay < bestWorstAnycastDelay
        bestWorstAnycastDelay = worstAnycastDelay;
        bestAnycastNodes = anycastNodes;
        bestRoundTripDelays = roundTripDelays;
        bestMaxLoad = maxLoad;
    end
```

This change introduces a new evaluation metric that prioritizes latency for anycast traffic over overall network load distribution.

The optimization process successfully determines the **best pair of anycast nodes, [4, 12]**, which minimizes the worst round-trip delay for the anycast service. The **worst link-load was reduced to 76.60 Gbps** from the initial 98.20 Gbps, indicating efficient traffic distribution across the network. The evaluation also provided insights into the round-trip delays for the services in the network. These metrics, including the worst and average round-trip delays for each service, are summarized in Table 2.4.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.04	5.42
2	11.07	5.83
3	4.42	2.90

Table 2.4: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [4 12]

2.5 Exercise: 1e

Required: Again, consider that you can freely select the anycast nodes of the anycast service. Try all possible combinations of 2 nodes and select the one that minimizes the average round-trip delay of the anycast service. Indicate the two selected nodes, the worst round-trip delay and the average round trip delay of each service.

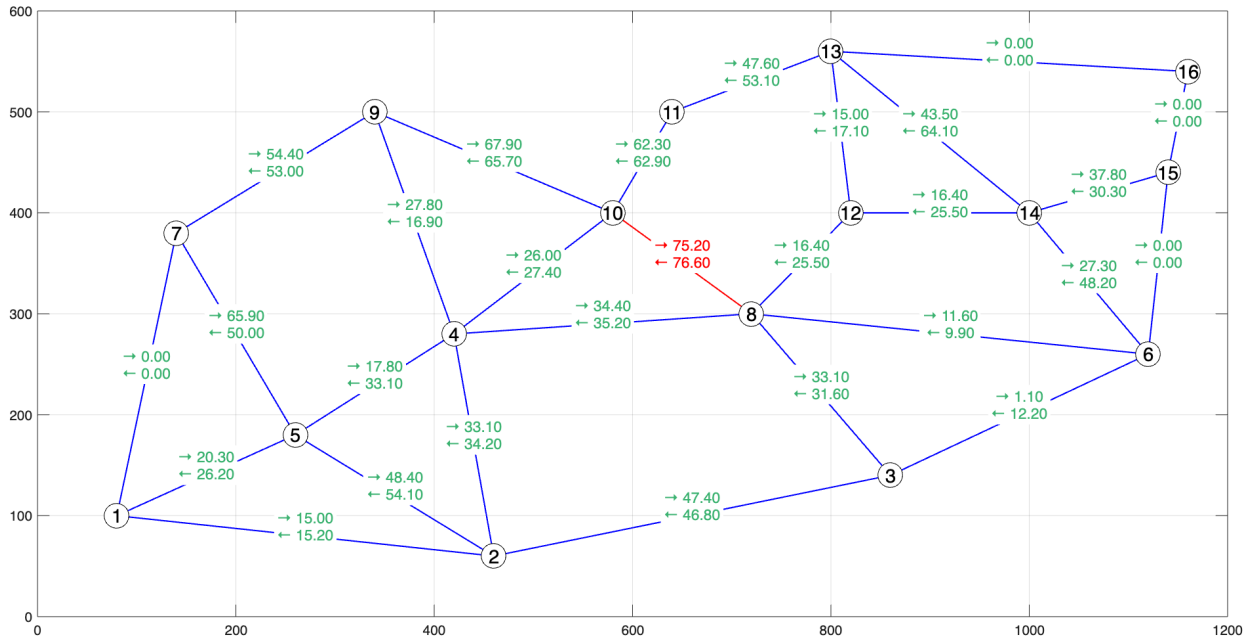

Figure 2.4: Visual representation of the Link Loads, using the nodes [5 14] as anycast nodes, where → represents source to destination and the ← destination to source

Listing 2.9: MATLAB code of the Optimization of Anycast Nodes and AVERAGE round-trip delay

```
for n1 = 1:nNodes-1
    for n2 = n1+1:nNodes
        anycastNodes = [n1, n2];
        Taux = zeros(nFlows, 4);
        roundTripDelays = zeros(1, nFlows);
        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), 1);
                sP{f} = shortestPath;
                nSP(f) = length(totalCost);
                Taux(f,:) = T(f,2:5);
```

```
                roundTripDelays(f) = 2 * totalCost * 1000;
            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
                    Taux(f,:) = T(f,2:5);
                    minCost = inf;
                    for acNode = anycastNodes
                        [shortestPath, totalCost] = kShortestPath(D, T(f,2),
                            acNode, 1);
                        if totalCost < minCost
                            minCost = totalCost;
                            sP{f} = shortestPath;
                            nSP(f) = 1;
                            Taux(f,2) = acNode;
                        end
                    end
                    roundTripDelays(f) = 2 * minCost * 1000;
                end
            end
        end

        % Compute link loads
        sol = ones(1, nFlows);
        Loads = calculateLinkLoads(nNodes, Links, Taux, sP, sol);

        % Find worst link load
        maxLoad= max(max(Loads(:,3:4)));

        % Find worst anycast service round-trip delay
        anycastDelays = roundTripDelays(T(:,1) == 3);
        avgAnycastDelay = mean(anycastDelays);

        % Update best solution if current solution has lower worst anycast delay
        if avgAnycastDelay < bestAvgAnycastDelay
            bestAvgAnycastDelay = avgAnycastDelay;
            bestAnycastNodes = anycastNodes;
            bestRoundTripDelays = roundTripDelays;
            bestMaxLoad = maxLoad;
        end
    end
end
```

This code maintains the strategy implemented in Exercise 1d, where all pairs of nodes are evaluated as potential anycast nodes. The primary change in this exercise is the optimization criterion: instead of minimizing the maximum round-trip delay for the anycast service (anycastDelays), the focus shifts to minimizing the **average round-trip delay**.

Listing 2.10: MATLAB code to store the best pair of nodes that minimize the average round- trip delay of the anycast service

```
1
2       avgAnycastDelay = mean(anycastDelays);
3
4       if avgAnycastDelay < bestAvgAnycastDelay
5           bestAvgAnycastDelay = avgAnycastDelay;
6           bestAnycastNodes = anycastNodes;
7           bestRoundTripDelays = roundTripDelays;
8           bestMaxLoad = maxLoad;
9       end
```

As we can see on the Listing 2.10 instead of focusing on the maximum delay for the anycast service (anycastDelays), this exercise focuses on the average delay (AVGANYCASTDELAY). Then the solution is updated if the current average anycast delay is lower than the previously observed best.

The optimization process successfully determines the **best pair of anycast nodes, [5, 14]**, which minimizes the AVERAGE ROUND-TRIP DELAY FOR THE ANYCAST SERVICE. The **worst link-load was reduced to 76.60 Gbps** from the initial 98.20 Gbps, indicating efficient traffic distribution across the network. The evaluation also provided insights into the round-trip delays for the services in the network. These metrics, including the worst and average round-trip delays for each service, are summarized in Table 2.5.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.04	5.42
2	11.07	5.83
3	4.90	2.52

Table 2.5: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [5 14]

2.6 Exercise: 1f

Required: Compare the results obtained in all experiments of Task 1 and draw all possible conclusions.

Across all experiments, the results for **Unicast Services** remain consistent for the WORST ROUND-TRIP DELAY and AVERAGE ROUND-TRIP DELAY. This consistency suggests that unicast service delays are unaffected by changes in node configurations or optimization criteria. Moreover, this stability indicates that optimizations for anycast services do not negatively impact unicast traffic, ensuring reliable performance for unicast flows.

For the **Anycast Service**, the results vary significantly across experiments, as shown in Table 2.6.

Experiment	**Worst Link Load (Gbps)**	**Best Anycast Nodes**	**Worst Round-Trip Delay (ms)**	**Average Round-Trip Delay (ms)**
1.a)	98.20	[3, 10]	6.16	3.43
1.c)	76.60	[1, 6]	6.41	3.02
1.d)	76.60	[4, 12]	4.42	2.90
1.e)	76.60	[5, 14]	4.90	2.52

Table 2.6: Table displaying the results for the experiments showing the key metrics WORST LINK-LOAD, WORST ROUND-TRIP DELAY, AVERAGE ROUND-TRIP DELAY and the chosen ANYCAST NODES. The values underlined in red color represent the criteria that we minimized for each experiment.

Experiments **c**, **d**, and **e** demonstrate significant improvements in both link utilization and round-trip delays for anycast services compared to **a**. Notably, the WORST LINK-LOAD stabilizes at **76.60 Gbps** in the optimization experiments, which indicates that it may have reached optimal value in the network topology. Figure 2.2, Figure 2.3, Figure 2.4 consistently show that the WORST LINK-LOAD occurs between nodes [8, 10] in the $8 \rightarrow 10$ direction, this highlights that this link is the critical bottleneck in the network, even with this 3 optimizations.

Furthermore, the selection of ANYCAST NODES significantly impacts delay metrics. The results demonstrate that strategic node placement in latency-sensitive networks can achieve better performance for both WORST ROUND-TRIP DELAY and AVERAGE ROUND-TRIP DELAY, depending on the chosen anycast nodes.

3. Task 2

3.1 Exercise: 2a

Required: Run the algorithm for 30 seconds with $k = 6$ candidate paths for each unicast flow when the anycast nodes of the anycast service are network nodes 3 and 10. Indicate the worst round-trip delay and the average round trip delay of each service (presenting all values in milliseconds) and the worst link load of the network. Indicate also the algorithm performance parameters.

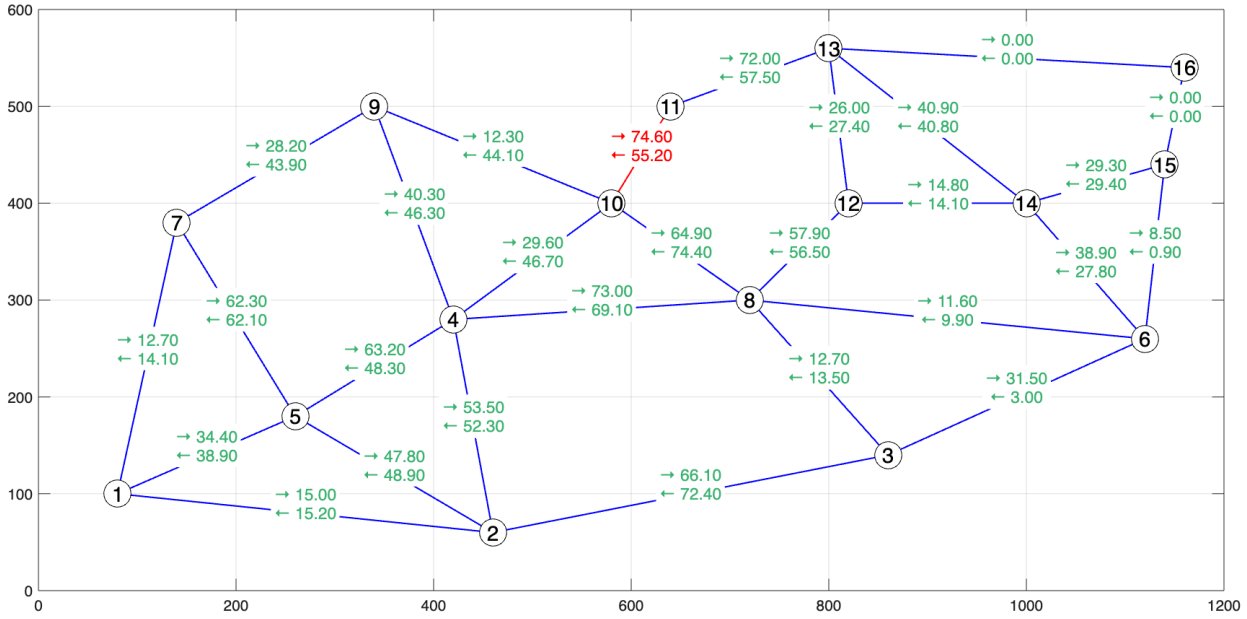

Figure 3.1: Visual representation of the Link Loads, using the nodes [3 10] as anycast nodes, where → represents source to destination and the ← destination to source

Listing 3.1: MATLAB code to minimizing the Worst Link-Load of the network using Multi Start Hill Climbing algorithm with initial Greedy Randomized solutions and with a stopping criterion defined by a given running time

```
1 load('InputDataProject2.mat')  % Load input data
```

```
2
3  % Initialize network parameters
4  nNodes = size(Nodes, 1);
5  nLinks = size(Links, 1);
6  nFlows = size(T, 1);
7
8  % Propagation velocity
9  v = 2e5;
10 D = L/v; % Propagation delay for each link
11
12 anycastNodes = [3, 10];  % Anycast nodes
13
14 % Initialize result storage variables
15 bestWll = 0;
16 bestAnycastNodes = anycastNodes;
17 bestRoundTripDelays = zeros(1, nFlows);
18 bestSolutions = cell(1, nFlows);
19 roundTripDelays = zeros(1, nFlows);
20
21 timeLimit = 30; % Time limit for hill climbing algorithm
22 k = 6; % Number of shortest paths to consider
23
24 sP = cell(1, nFlows);
25 nSP = zeros(1, nFlows);
26
27 for f = 1:nFlows     % Process each flow
28     if T(f, 1) == 1 || T(f, 1) == 2 % UNICAST SERVICE
29         % Find k shortest paths
30         [shortestPaths, totalCosts] = kShortestPath(D, T(f,2), T(f,3), k);
31
32         % Store paths and number of paths
33         sP{f} = shortestPaths;
34         nSP(f) = length(totalCosts);
35         Taux(f,:) = T(f,2:5);
36
37     elseif T(f, 1) == 3 % ANYCAST SERVICE
38         if ismember(T(f,2), anycastNodes)
39             % Source is already an anycast node
40             sP{f} = {T(f,2)};
41             nSP(f) = 1;
42             Taux(f,:) = T(f,2:5);
43             Taux(f,2) = T(f,2); % ---> origin node
44
45         else
46             % Find the nearest anycast node
47             Taux(f,:) = T(f,2:5);
```

```
            minCost = inf;

            for acNode = anycastNodes
                [shortestPath, totalCost] = kShortestPath(D, T(f,2), acNode, 1);
                if totalCost < minCost
                    minCost = totalCost;
                    sP{f} = shortestPath;
                    nSP(f) = 1;
                    Taux(f,2) = acNode;
                    bestAnycastNode = acNode;
                end
            end
        end
    end
end

% Run Hill Climbing Optimization
fprintf('Running Hill Climbing Optimization...\n');
[bestSolCycle, bestSol, bestObjective, noCycles, avObjective, bestTime] = ...
    HillClimbGreedyAlgorithm(nNodes, Links, Taux, sP, nSP, timeLimit);

% Hill climbing Statistics
fprintf('total  n   of cycles: %d\n', noCycles);
fprintf('Time of the best Solution: %.2f ms\n', bestTime*1000);
fprintf('Cycle of the best Solution: %d\n', bestSolCycle);
fprintf('Best Objective (Loads): %.2f Gbps\n', bestObjective);
fprintf('Average Objective: %.2f Gbps\n', avObjective);

solutions = cell(1, nFlows);

for p = 1:nFlows
    bestPathIndex = bestSol(p);
    solutions{p} = sP{p}{bestPathIndex}; % Note the double {} since sP is a cell
        array of cell arrays
end

% Initialize oneWayDelays to store the delay for each solution path
oneWayDelays = zeros(1, length(solutions));

% Iterate through each path in the solutions array
for i = 1:length(solutions)
    if ~isempty(solutions{i}) % Check if the solution is not empty
        oneWayDelays(i) = calculatePathDelay(solutions{i}, D);
    else
        oneWayDelays(i) = Inf;
    end
```

```
end

% Initialize arrays to store round-trip delays for each service
s1rtd = []; % For service type 1
s2rtd = []; % For service type 2
s3rtd = []; % For service type 3

% Loop through all flows
for p = 1:nFlows
    roundTripDelay = oneWayDelays(p) * 2000; % Calculate round-trip delay in ms
    if T(p, 1) == 1 % For service type 1
        s1rtd = [s1rtd, roundTripDelay];
    elseif T(p, 1) == 2  % For service type 2
        s2rtd = [s2rtd, roundTripDelay];
    elseif T(p, 1) == 3 % For service type 3
        s3rtd = [s3rtd, roundTripDelay];
    end
end

fprintf("\n-----------------------| Unicast Service S = 1\n")
fprintf("Worst round-trip delay  \t= %.2f ms \n", max(s1rtd));
fprintf("Average round-trip delay  \t= %.2f ms \n", mean(s1rtd));
fprintf("\n")
fprintf("\n-----------------------| Unicast Service S = 2\n")
fprintf("Worst round-trip delay  \t= %.2f ms \n", max(s2rtd));
fprintf("Average round-trip delay  \t= %.2f ms \n", mean(s2rtd));
fprintf("\n")
fprintf("\n-----------------------| Anycast Service S = 3\n")
fprintf("Worst round-trip delay  \t= %.2f ms \n", max(s3rtd));
fprintf("Average round-trip delay  \t= %.2f ms \n", mean(s3rtd));
fprintf("\n")

% Compute and plot the link loads using the optimized solution
Loads = calculateLinkLoads(nNodes, Links, Taux, sP, bestSol);
plotGraphWithLoadsDynamicColor(Nodes, Links, Loads, 2); % Plot graph with loads
```

The code implemented for Task 2 addresses the optimization problem of computing a single routing path for each traffic flow of both unicast services ($S = 1$ and $S = 2$), aiming to minimize the *Worst Link-Load* of the network.

Once the paths for all flows are determined, the **Hill Climbing Algorithm** is applied:

- The algorithm begins with a set of **random greedy solutions**.
- It dynamically calculates and balances the network's link loads by iteratively selecting flows and adjusting their paths to minimize the *Worst Link-Load*.

- The Hill Climbing process evaluates the effect of each selected path on the overall load, ensuring an optimized solution.

This approach dynamically considers the *Worst Link-Load* at every iteration, leading to a more balanced network configuration.

After obtaining the optimized routing paths:

- For each flow, the path chosen by the Hill Climbing algorithm is extracted.
- Using the function `calculatePathDelay` (Listing 3.2), the delay for each selected path is computed.

The delays are then categorized based on the service type ($S = 1$, $S = 2$, and $S = 3$), enabling detailed analysis of round-trip delays for each service.

Listing 3.2: MATLAB code to calculate the delay for each link in a chosen path

```
function delay = calculatePathDelay(path, netCostMatrix)
    % Initialize delay
    delay = 0;

    % Loop through each link in the path
    for i = 1:length(path) - 1
        % Extract nodes for the current link
        node1 = path(i);
        node2 = path(i + 1);

        % Add the cost of this link to the total delay
        delay = delay + netCostMatrix(node1, node2);
    end
end
```

The final steps of the code involve:

- **Statistics:** The results include:
  - Worst and average round-trip delays for each service.
  - The overall *Worst Link-Load* and other algorithm performance metrics (e.g., number of cycles and time to obtain best solution).

  These statistics provide insights into the efficiency of the Hill Climbing algorithm and the network's performance under the optimized routing configuration.

- **Graph Visualization:** The function `plotGraphWithLoadsDynamicColor` (Listing 3.3) is used to visualize the network's link loads.
  - The visualization highlights link with the highest loads in red.

Listing 3.3: MATLAB code to print the graph with loads on links between the nodes

```
function plotGraphWithLoadsDynamicColor(Nodes, Links, Loads, n)
    % Plot the graph with dynamic coloring for loads and links
    figure(n);
    co = Nodes(:,1) + 1j * Nodes(:,2); % Complex coordinates
    plot(co, 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'w', 'MarkerSize',
        18);
    hold on;

    % Find the highest load value and its index
    [maxLoad, maxIndex] = max(max(Loads(:,3:4), [], 2));

    % Plot links and annotate with load values
    for i = 1:size(Links,1)
        % Determine line color based on the link's load
        if i == maxIndex
            lineColor = 'r'; % Red for the worst link
        else
            lineColor = 'b'; % Light blue for normal links
        end

        % Plot the link with the determined color
        plot([Nodes(Links(i,1),1) Nodes(Links(i,2),1)], [Nodes(Links(i,1),2) Nodes
            (Links(i,2),2)], 'Color', lineColor, 'LineWidth', 1);

        % Calculate the midpoint of the link
        midX = (Nodes(Links(i,1),1) + Nodes(Links(i,2),1)) / 2;
        midY = (Nodes(Links(i,1),2) + Nodes(Links(i,2),2)) / 2;

        % Determine text color for the loads
        if Loads(i,3) == maxLoad || Loads(i,4) == maxLoad
            textColor = 'r'; % Red for the highest value
        else
            textColor = '#3CB371'; % Dark green for normal values
        end

        % Display the link loads at the midpoint with white background
        text(midX, midY + 10, ...
             sprintf('    %.2f\ n   %.2f', Loads(i,3), Loads(i,4)), ...
```

```
                'HorizontalAlignment', 'center', ...
                'Color', textColor, ...
                'FontWeight', 'normal', ...
                'FontSize', 11, ...
                'BackgroundColor', 'w', ... % White background for better visibility
                'Margin', 2); % Add some padding
    end

    % Re-plot nodes for visibility
    plot(co, 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'w', 'MarkerSize',
        18);

    % Label the nodes
    for i = 1:length(co)
        text(Nodes(i,1), Nodes(i,2), sprintf('%d', i), 'HorizontalAlignment', '
            center', 'Color', 'k', 'FontSize', 14);
    end

    grid on;
    hold off;
end
```

The code for Task 2 effectively implements:

- Path computation for each flow based on service type.
- Optimization of network load using the Hill Climbing Algorithm with random greedy solutions, equal to the one made on the pratical classrooms.
- Detailed delay and load statistics for analysis.
- Dynamic visualization of link loads for enhanced understanding.

This approach ensures that the routing paths selected are not only efficient but also result in a balanced network configuration.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.40	6.59
2	11.07	6.16
3	6.16	3.43

Table 3.1: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [3 10]

Total nº of cycles:	6740
Time of the best Solution(ms):	400.59
Cycle of Best solution(ms):	57
Best Objective(Gbps):	74.60
Average Objective(Gbps):	78.58

Table 3.2: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [3 10]

3.2 Exercise: 2b

Required: Repeat experiment 2.a with the anycast nodes selected in experiment 1.c.

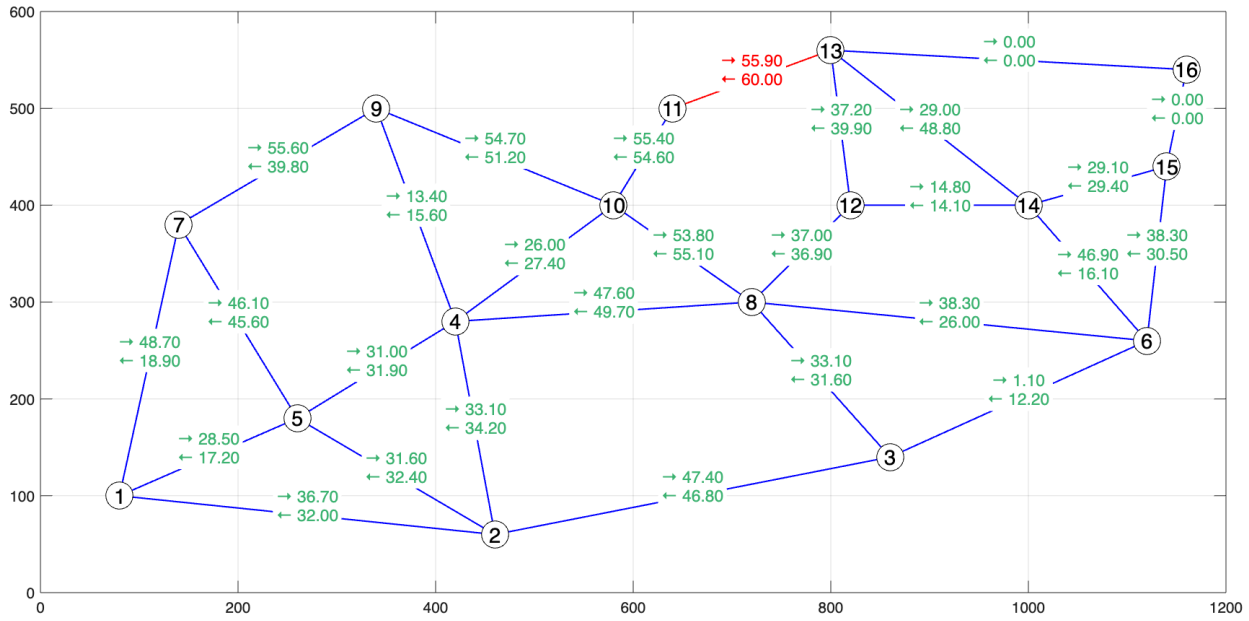

Figure 3.2: Visual representation of the Link Loads, using the nodes [1 6] as anycast nodes, where → represents source to destination and the ← destination to source

The code for this experiment is identical to the one used in **Task 2a**, with the only modification being the change in the anycast nodes. Instead of the initial configuration of $[3, 10]$, the anycast nodes are now set to $[1, 6]$. This adjustment impacts the path selection for the anycast service ($S = 3$), while all other logic and operations in the code remain unchanged.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.76	5.58
2	12.00	6.90
3	6.41	3.02

Table 3.3: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [1 6]

Total nº of cycles:	5805
Time of the best Solution(ms):	89.44
Cycle of Best solution(ms):	9
Best Objective(Gbps):	60.00
Average Objective(Gbps):	63.72

Table 3.4: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [1 6]

3.3 Exercise: 2c

Required: Repeat experiment 2.a with the anycast nodes selected in experiment 1.d.

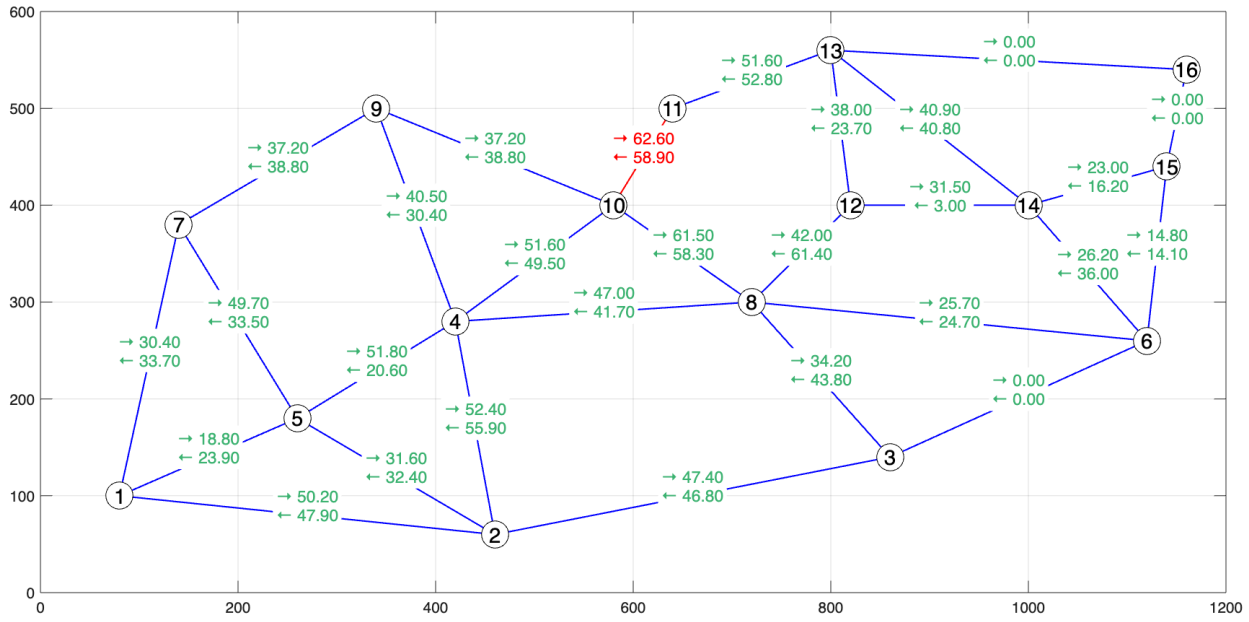

Figure 3.3: Visual representation of the Link Loads, using the nodes [4 12] as anycast nodes, where → represents source to destination and the ← destination to source

The code for this experiment is identical to the one used in **Task 2a**, with the only modification being the change in the anycast nodes. Instead of the initial configuration of $[3, 10]$, the anycast nodes are now set to $[4, 12]$. This adjustment impacts the path selection for the anycast service ($S = 3$), while all other logic and operations in the code remain unchanged.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	9.04	6.48
2	11.07	6.34
3	4.42	2.90

Table 3.5: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [4 12]

Total nº of cycles:	4529
Time of the best Solution(ms):	44.59
Cycle of Best solution(ms):	4
Best Objective(Gbps):	62.60
Average Objective(Gbps):	66.07

Table 3.6: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [4 12]

3.4 Exercise: 2d

Required: Repeat experiment 2.a with the anycast nodes selected in experiment 1.e.

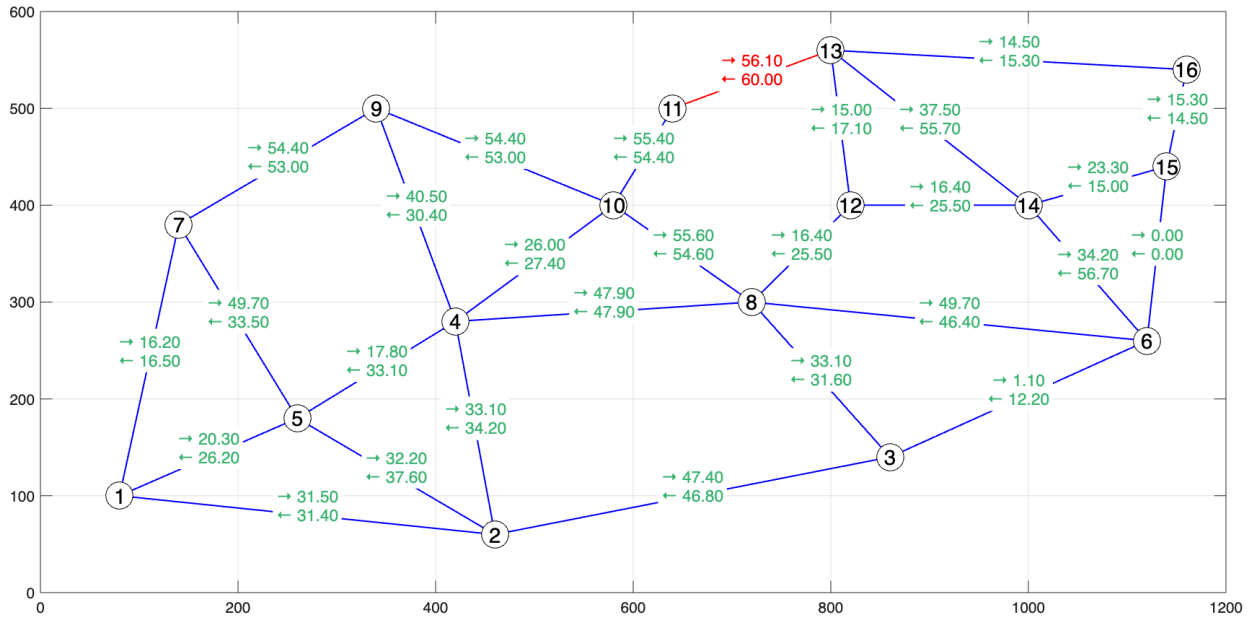

Figure 3.4: Visual representation of the Link Loads, using the nodes [5 14] as anycast nodes, where → represents source to destination and the ← destination to source

The code for this experiment is identical to the one used in **Task 2a**, with the only modification being the change in the anycast nodes. Instead of the initial configuration of $[3, 10]$, the anycast nodes are now set to $[5, 14]$. This adjustment impacts the path selection for the anycast service ($S = 3$), while all other logic and operations in the code remain unchanged.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	10.13	6.38
2	11.07	6.59
3	4.90	2.52

Table 3.7: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [5 14]

Total nº of cycles:	5417
Time of the best Solution(ms):	141.64
Cycle of Best solution(ms):	14
Best Objective(Gbps):	60.00
Average Objective(Gbps):	64.16

Table 3.8: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [5 14]

3.5 Exercise: 2e

Required: Compare the results obtained in all experiments of this Task 2 and of the previous Task 1 and draw all meaningful conclusions both concerning the differences between the solutions and the differences between the performance of the algorithm.

Analyzing the results presented in the tables, the solutions from **Task 2**, Table 3.1 Table 3.3 Table 3.5 Table 3.7, outperform those from **Task 1**,Table 2.1 Table 2.3, Table 2.4 Table 2.5, in link load optimization, as expected. This was anticipated because in Task 2, we use the Hill Climbing Algorithm with Greedy Randomized solutions to select the path configuration that minimizes the worst link-load.

In Task 2, we aim to minimize the worst link-load by selecting paths from a set of six candidates generated for each Unicast Service (S = 1 and S = 2). By randomly choosing flows to calculate the loads, the algorithm balances the network by considering the worst link-load dynamically, **based on already-selected flows**. However, we could possibly achieve better results by modifying the approach.

Specifically, instead of using a Hill Climbing Optimization with random greedy solutions, we could first select the best paths for the Anycast Service (S = 3). Considering this service has only

one possible path, pre-selecting its paths would allow subsequent flow selections to better balance the network.

The improvements on link load optimization come with a trade-off: in **Task 1**, the values show slightly lower round-trip delays, due to more direct paths being chosen without algorithmic optimization for link load balancing.

After analyzing the results, the best configuration was achieved with **anycast nodes [5, 14]**, resulting in:

- **Worst Link-Load**: 60 Gbps (the lowest observed across all configurations).
- Better results for **Worst Round-trip delay and Average Round-trip delays** for the 3 services.

The efficiency, across all experiments Table 3.2 Table 3.4 Table 3.6 Table 3.8, of the Hill Climbing algorithm is evident, as it obtains the best results in relatively few cycles into the search for the best path selection.

4. Task 3

4.1 Exercise: 3a

Required: Run the algorithm for 60 seconds with $k = 12$ candidate paths for each traffic flow of unicast service ($S = 1$) and with $k = 12$ candidate pairs of link disjoint paths for each traffic flow of unicast service ($S = 2$). Consider that the anycast nodes of the anycast service are network nodes 3 and 10. Indicate the worst round-trip delay and the average round trip delay of each service1 (presenting all values in milliseconds) and the worst link load of the network. Indicate also the algorithm performance parameters.

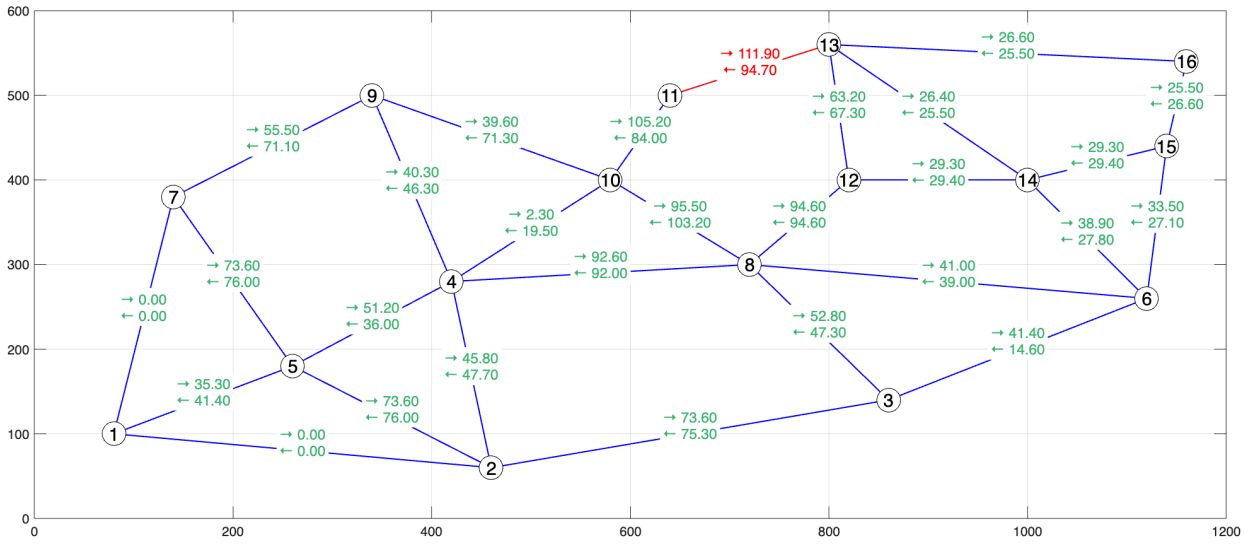

Figure 4.1: Visual representation of the Link Loads, using the nodes [3 10] as anycast nodes, where → represents source to destination and the ← destination to source

Listing 4.1: MATLAB code to minimizing the worst link-load of the network (considering the load of a link given by the load of the working paths plus the reservation for the backup paths) using a Multi Start Hill Climbing algorithm with initial Greedy Randomized solutions and with a stopping criterion defined by a given running time. Besides the best solution found the algorithm must also output the same performance parameters as defined in Task 2.

```
% Load input data
load('InputDataProject2.mat')

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

% Propagation velocity
v = 2e5;
D = L/v; % Propagation delay for each link

anycastNodes = [5, 14];  % Anycast nodes

% Initialize result storage variables
bestWll = 0;
bestAnycastNodes = anycastNodes;
bestRoundTripDelays = zeros(1, nFlows);
bestSolutions = cell(1, nFlows);
roundTripDelays = zeros(1, nFlows);

timeLimit = 60; % Time limit for hill climbing algorithm
k = 12; % Number of shortest paths to consider

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


for f = 1:nFlows    % Process each flow
    if T(f, 1) == 1 % UNICAST SERVICE
        [shortestPaths, totalCosts] = kShortestPath(D, T(f,2), T(f,3), k);
        sP{1,f} = shortestPaths;
        sP{2,f} = 0;
        nSP(f) = length(totalCosts);
        Taux(f,:) = T(f,2:5);
    elseif T(f,1) == 2 % UNICAST SERVICE
        [firstPaths, secondPaths, totalPairCosts] = kShortestPathPairs(D, T(f,2),
            T(f,3), k);
        sP{1, f} = firstPaths;  % Working paths
        sP{2, f} = secondPaths; % Protection paths
        nSP(f) = length(totalPairCosts); % Number of valid path pairs
        Taux(f,:) = T(f,2:5);
    elseif T(f, 1) == 3 % ANYCAST SERVICE
        if ismember(T(f,2), anycastNodes)
            % Source is already an anycast node
            sP{1,f} = {T(f,2)};
            sP{2,f} = 0;
```

```
            nSP(f) = 1;
            Taux(f,:) = T(f,2:5);
            Taux(f,2) = T(f,2); % ---> original node
        else
            % Find the nearest anycast node
            Taux(f,:) = T(f,2:5);
            minCost = inf;

            for acNode = anycastNodes
                [shortestPath, totalCost] = kShortestPath(D, T(f,2), acNode, 1);
                if totalCost < minCost
                    minCost = totalCost;
                    sP{1,f} = shortestPath;
                    sP{2,f} = 0;
                    nSP(f) = 1;
                    Taux(f,2) = acNode;
                    bestAnycastNode = acNode;
                end
            end
        end
    end
end

% Run Hill Climbing Optimization
fprintf('Running Hill Climbing Optimization...\n');
[bestSolCycle, bestSol, bestObjective, noCycles, avObjective, bestTime] = ...
    HillClimbGreedyAlgorithm(nNodes, Links, Taux, sP, nSP, timeLimit);

% Debug print hill climbing results
fprintf('total  n   of cycles: %d\n', noCycles);
fprintf('Time of the best Solution: %.2f ms\n', bestTime*1000);
fprintf('Cycle of the best Solution: %d\n', bestSolCycle);
fprintf('Best Objective (Loads): %.2f Gbps\n', bestObjective);
fprintf('Average Objective: %.2f Gbps\n', avObjective);

solutions = cell(1, nFlows);

for p = 1:nFlows
    bestPathIndex = bestSol(p);
    solutions{p} = sP{1,p}{bestPathIndex}; % Note the double {} since sP is a cell
        array of cell arrays
end

% Initialize oneWayDelays to store the delay for each solution path
oneWayDelays = zeros(1, length(solutions));

```

```
% Iterate through each path in the solutions array
for i = 1:length(solutions)
    % Check if the solution is not empty
    if ~isempty(solutions{i})
        % Calculate the delay for the current path
        oneWayDelays(i) = calculatePathDelay(solutions{i}, D);
    else
        % If the solution is empty, set the delay to Inf or 0
        oneWayDelays(i) = Inf; % or 0, depending on how you want to handle empty
            paths
    end
end

% Initialize arrays to store round-trip delays for each service
s1rtd = []; % For service type 1
s2rtd = []; % For service type 2
s3rtd = []; % For service type 3

% Loop through all flows
for p = 1:nFlows
    roundTripDelay = oneWayDelays(p) * 2000; % Calculate round-trip delay in ms

    if T(p, 1) == 1
        % For service type 1
        s1rtd = [s1rtd, roundTripDelay];
    elseif T(p, 1) == 2
        % For service type 2
        s2rtd = [s2rtd, roundTripDelay];
    elseif T(p, 1) == 3
        % For service type 3
        s3rtd = [s3rtd, roundTripDelay];
    end
end


fprintf("\n----------------------| Unicast Service S = 1\n")
fprintf("Worst round-trip delay  \t= %.2f ms \n", max(s1rtd));
fprintf("Average round-trip delay  \t= %.2f ms \n", mean(s1rtd));
fprintf("\n")
fprintf("\n----------------------| Unicast Service S = 2\n")
fprintf("Worst round-trip delay  \t= %.2f ms \n", max(s2rtd));
fprintf("Average round-trip delay  \t= %.2f ms \n", mean(s2rtd));
fprintf("\n")
fprintf("\n----------------------| Anycast Service S = 3\n")
fprintf("Worst round-trip delay  \t= %.2f ms \n", max(s3rtd));
fprintf("Average round-trip delay  \t= %.2f ms \n", mean(s3rtd));
```

```
fprintf("\n")

% Compute and plot the link loads using the optimized solution
Loads = calculateLinkLoadsPairs(nNodes, Links, Taux, sP, bestSol);
plotGraphWithLoadsDynamicColor(Nodes, Links, Loads, 2); % Plot graph with loads
fprintf('Graph with dynamic loads plotted.\n');
```

The code for Task 3 solves the optimization problem of computing routing paths for all traffic flows while minimizing the *Worst Link-Load* of the network. In addition, it incorporates a **protection mechanism** (1:1 link-disjoint routing paths) for unicast service $S = 2$.

The main loop processes each flow based on its service type:

1. **Unicast Services ($S = 1$):**
   - For each flow, the code computes the **12 shortest paths** using the `KShortestPath` function.
   - These paths are stored for later use in the optimization process.
2. **Unicast Services with Protection ($S = 2$):**
   - For each flow, the code computes **12 pairs of link-disjoint paths** using the `KShortestPathPairs` function.
   - Each pair includes a **working path** and a **protection path**. The protection path is reserved to ensure reliability in case of a link failure.
   - The number of valid path pairs is stored for further use.
3. **Anycast Service ($S = 3$):**
   - The code selects the **nearest node** from the predefined anycast node set.
   - If the source node is already an anycast node, it directly uses that node as the destination.
   - Otherwise, the shortest path to the nearest anycast node is computed and stored.

Once the paths for all flows are determined, the **Hill Climbing Algorithm** is applied:

- The algorithm starts with a set of **random greedy solutions**.
- It iteratively adjusts the selected paths for each flow to minimize the *Worst Link-Load*.
- For $S = 2$, the optimization accounts for both the **working path** and the **protection path** in the load calculation.
- The algorithm outputs:

– The **best solution** with the lowest worst link-load.

– Performance metrics, including the number of cycles, time to the best solution, and the cycle where the best solution was found.

In order to fulfill what is asked "CONSIDERING THE LOAD OF A LINK GIVEN BY THE LOAD OF THE WORKING PATHS PLUS THE RESERVATION FOR THE BACKUP PATHS" we must alter the function CalculateLinkLoads inside of the Hill Climb Algorithm, Listing 4.2.

Listing 4.2: MATLAB code of Hill Climbing Optimization with random greedy solutions done in Task9 of the pratical classes

```
function [bestSolCycle, bestSol, bestObjective, noCycles, avObjective, bestTime]
    = HillClimbGreedyAlgorithm(nNodes, Links, T, sP, nSP, timeLimit)
    t = tic; % Start timer
    nFlows = size(T, 1); % Number of flows
    bestObjective = inf; % Initialize best load to infinity
    noCycles = 0; % Counter for the number of solutions generated
    aux = 0; % Cumulative load for averaging
    bestTime = 0; % Variable to store the time when the best load was found
    bestSolCycle = 0;

    while toc(t) < timeLimit
        % ------------------------- INITIAL SOLUTION SELECTION
            -------------------------
        % [RANDOM GREEDY ALGORITHM]
        %
        sol = zeros(1, nFlows); % Initialize the solution to zero
        randFlows = randperm(nFlows); % Randomize the order of flows
        for flow = randFlows
            path_index = 0; % Initialize the best path index
            best_load = inf; % Initialize the best load for this flow
            for path = 1:nSP(flow)
                sol(flow) = path; % Assign the current path
                Loads = calculateLinkLoadsPairs(nNodes, Links, T, sP, sol); %
                    Calculate the link loads
                load = max(max(Loads(:, 3:4))); % Evaluate the maximum load

                % Update the best path if the load is better
                if load < best_load
                    path_index = path;
                    best_load = load;
                end
            end
            sol(flow) = path_index; % Assign the best path found for this flow
        end
```

```
32
33          % Calculate the initial load after the greedy solution
34          Loads = calculateLinkLoadsPairs(nNodes, Links, T, sP, sol);
35          load = max(max(Loads(:, 3:4)));
36
37          % ------------------------- HILL CLIMBING OPTIMIZATION
                -------------------------
38          % [HILL CLIMBING ALGORITHM]
39          %
40          improved = true;
41          while improved
42              bestLocalLoad = load;
43              bestLocalSol = sol;
44
45              % exploring "neighboring" solutions
46              % process continues until no further improvements can be found
47              %
48              for flow = 1:nFlows
49                  for path = 1:nSP(flow)
50                      if path ~= sol(flow)  % ---> Neighbor Exploration
51                          auxSol = sol;
52                          auxSol(flow) = path;
53                          % ---> Calculate the load for the new solution
54                          Loads = calculateLinkLoadsPairs(nNodes, Links, T, sP,
                                auxSol);
55                          auxLoad = max(max(Loads(:, 3:4)));
56
57                          % Check if the new load is better
58                          if auxLoad < bestLocalLoad
59                              bestLocalLoad = auxLoad;
60                              bestLocalSol = auxSol;
61                          end
62                      end
63                  end
64              end
65
66              % Update solution if a better local solution is found
67              if bestLocalLoad < load
68                  load = bestLocalLoad;
69                  sol = bestLocalSol;
70              else
71                  improved = false; % ---> Stop if no improvement is found
72              end
73          end
74
75          % ---> Track the Best Solution Found
```

```
76          noCycles = noCycles + 1;
77          aux = aux + load;
78
79          if load < bestObjective
80              bestSol = sol;
81              bestObjective = load;
82              bestTime = toc(t); % ---> Track the time when the best load was found
83              bestSolCycle = noCycles;
84          end
85      end
86
87      avObjective = aux / noCycles;
88  end
```

The function `CalculateLinkLoadsPairs` (Listing 4.3) is designed to calculate the cumulative loads on the network's links, incorporating the working paths and protection paths specifically for service $S = 2$. This function takes advantage of the structure of `sP`, which now contains two paths for each flow of service $S = 2$: a **working path** and a **protection path**. For services $S = 1$ and $S = 3$, the second path (`sP{2, i}`) is zero, ensuring these services are processed only with their single routing path.

The function iterates through all flows and performs the following steps:

- For the **working path**, it calculates the load on each link along the path in both forward and reverse directions. The load values are based on the flow demands specified in the input matrix `T`.
- For service $S = 2$, if a **protection path** exists, the function similarly calculates and adds the load on each link along the protection path. This ensures that the link loads include both the working and reserved protection path loads.
- The results are compiled into an output matrix `Loads`, where the forward and reverse loads for each link are stored.

This approach ensures that the cumulative load on the network is accurately calculated, taking into account the additional constraints introduced by the protection paths for service $S = 2$, while maintaining compatibility with services $S = 1$ and $S = 3$.

Listing 4.3: MATLAB code to calculate the loads form each link having in cosideration that service 2 is the cumulative working and protection path loads

```
1  function Loads = calculateLinkLoadsPairs(nNodes, Links, T, sP, Solution)
2      nFlows = size(T, 1); % Number of flows
3      nLinks = size(Links, 1); % Number of links
```

```
    aux = zeros(nNodes); % Auxiliary matrix to store link loads

    for i = 1:nFlows
        if Solution(i) > 0
            % Process working path
            workingPath = sP{1, i}{Solution(i)};
            for j = 2:length(workingPath)
                aux(workingPath(j - 1), workingPath(j)) = aux(workingPath(j - 1),
                    workingPath(j)) + T(i, 3); % Add load (forward direction)
                aux(workingPath(j), workingPath(j - 1)) = aux(workingPath(j),
                    workingPath(j - 1)) + T(i, 4); % Add load (reverse direction)
            end

            % Process backup path if it exists
            if ~isempty(sP{2, i}) && ~isequal(sP{2, i}, 0)
                backupPath = sP{2, i}{Solution(i)};
                for j = 2:length(backupPath)
                    aux(backupPath(j - 1), backupPath(j)) = aux(backupPath(j - 1),
                        backupPath(j)) + T(i, 3); % Add load (forward direction)
                    aux(backupPath(j), backupPath(j - 1)) = aux(backupPath(j),
                        backupPath(j - 1)) + T(i, 4); % Add load (reverse direction
                        )
                end
            end
        end
    end

    % Compile the loads into the output matrix
    Loads = [Links zeros(nLinks, 2)];
    for i = 1:nLinks
        Loads(i, 3) = aux(Loads(i, 1), Loads(i, 2)); % Load in forward direction
        Loads(i, 4) = aux(Loads(i, 2), Loads(i, 1)); % Load in reverse direction
    end
end
```

After obtaining the optimized solution:

- The selected path for each flow is extracted.
- Using the `calculatePathDelay` function, Listing 3.2 , the delay for each path is obtained.
- The delays are categorized based on the service type ($S = 1$, $S = 2$, and $S = 3$) to enable detailed analysis of round-trip delays.

The final part of the code calculates and visualizes the link loads:

1. **Link Load Calculation:**

- The `calculateLinkLoadsPairs` function, Listing 4.3, calculates the cumulative link loads, including both working and backup paths for $S = 2$.

2. **Graph Visualization:**

   - The `plotGraphWithLoadsDynamicColor` function, Listing 3.3, visualizes the network, highlighting:
     - The **Worst Link-Load** in red.

The code outputs the statistics as in **Task 2**:

1. **Worst and Average Round-Trip Delays:**

   - For each service ($S = 1$, $S = 2$, and $S = 3$).

2. **Worst Link-Load:**

   - Based on the optimized solution.

3. **Performance Metrics:**

   - Total number of cycles.
   - Time to find the best solution.
   - Cycle number of the best solution.

The Task 3 code builds upon the logic of Task 2 but introduces additional constraints for $S = 2$ by requiring link-disjoint protection paths. The **Hill Climbing Algorithm** dynamically balances the network by considering both working and protection paths, ensuring reliability while minimizing the *Worst Link-Load*. The resulting statistics and visualizations provide insights into the performance and efficiency of the routing optimization process.

Service	**Worst round-trip Delay(ms)**	**Average round-trip Delay(ms)**
1	11.27	6.72
2	11.07	6.15
3	6.16	3.43

Table 4.1: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [3 10]

Total nº of cycles:	4367
Time of the best Solution(ms):	37.80
Cycle of Best solution(ms):	1
Best Objective(Gbps):	111.90
Average Objective(Gbps):	114.60

Table 4.2: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [3 10]

4.2 Exercise: 3b

Required: Repeat experiment 3.a with the anycast nodes selected in experiment 1.c.

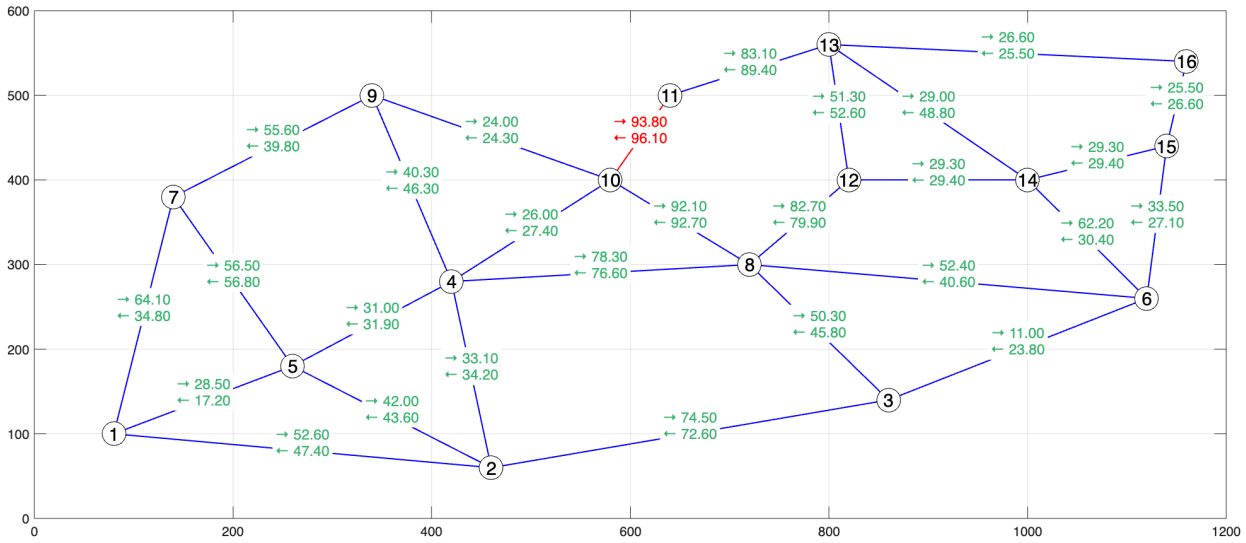

Figure 4.2: Visual representation of the Link Loads, using the nodes [1 6] as anycast nodes, where → represents source to destination and the ← destination to source

The code for this experiment is identical to the one used in **Task 3a**, with the only modification being the change in the anycast nodes. Instead of the initial configuration of $[3, 10]$, the anycast nodes are now set to $[1, 6]$.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	11.98	6.41
2	11.07	6.04
3	6.41	3.02

Table 4.3: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [1 6]

Total nº of cycles:	3940
Time of the best Solution(ms):	28.11
Cycle of Best solution(ms):	1
Best Objective(Gbps):	96.10
Average Objective(Gbps):	98.34

Table 4.4: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [1 6]

4.3 Exercise: 3c

Required: Repeat experiment 3.a with the anycast nodes selected in experiment 1.d.

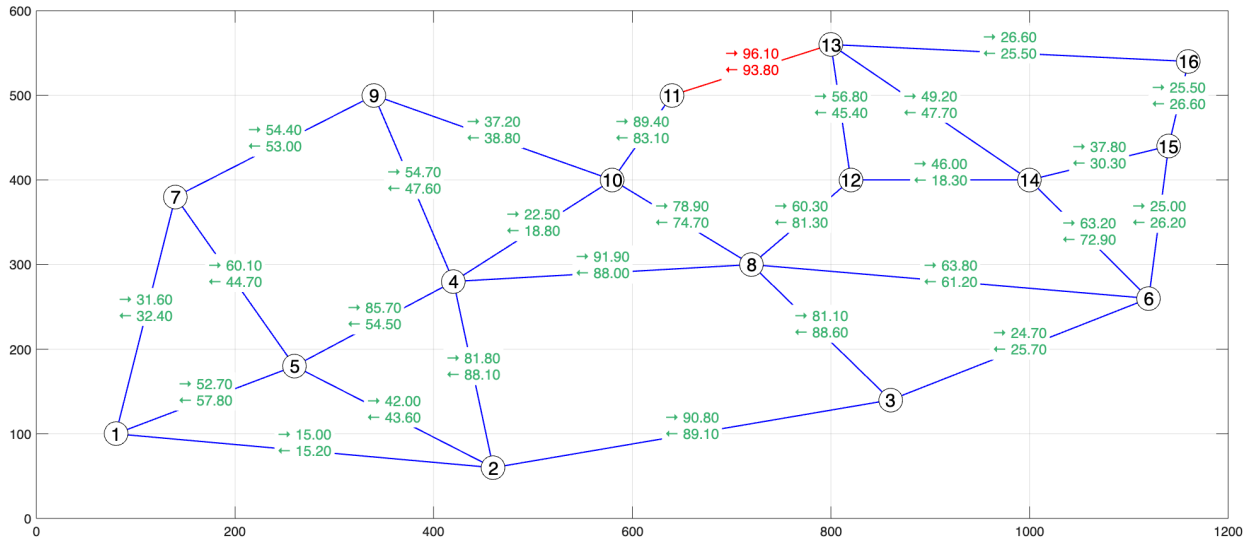

Figure 4.3: Visual representation of the Link Loads, using the nodes [4 12] as anycast nodes, where → represents source to destination and the ← destination to source

The code for this experiment is identical to the one used in **Task 3a**, with the only modification being the change in the anycast nodes. Instead of the initial configuration of $[3, 10]$, the anycast nodes are now set to $[4, 12]$.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	11.06	6.37
2	11.07	6.04
3	4.42	2.90

Table 4.5: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [4 12]

Total nº of cycles:	2871
Time of the best Solution(ms):	149.62
Cycle of Best solution(ms):	4
Best Objective(Gbps):	96.10
Average Objective(Gbps):	99.43

Table 4.6: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [4 12]

4.4 Exercise: 3d

Required: Repeat experiment 3.a with the anycast nodes selected in experiment 1.e.

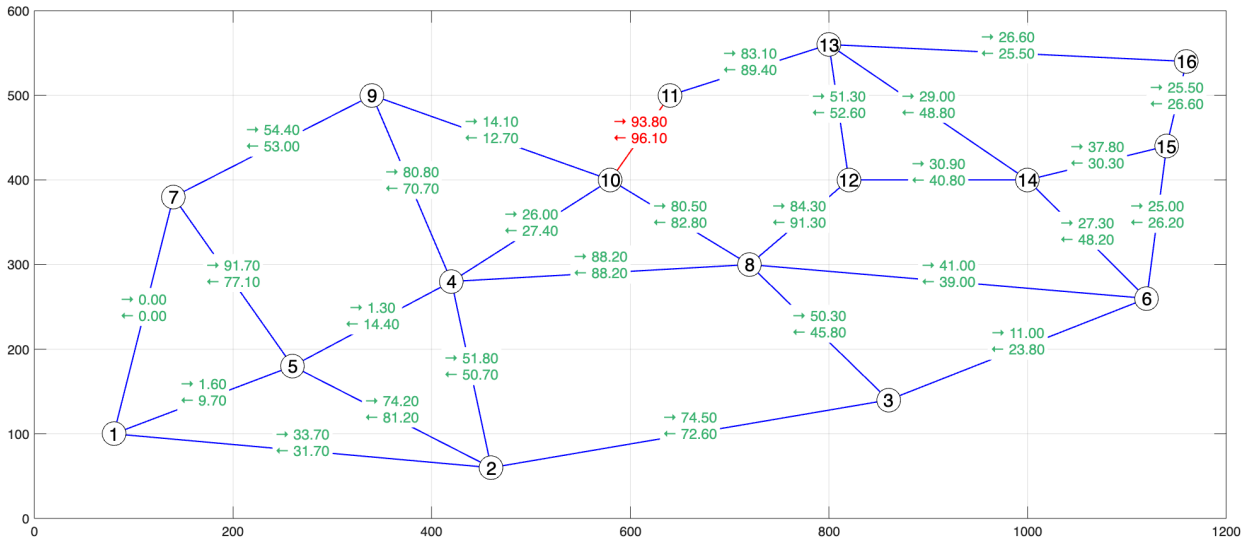

Figure 4.4: Visual representation of the Link Loads, using the nodes [5 14] as anycast nodes, where → represents source to destination and the ← destination to source

The code for this experiment is identical to the one used in **Task 3a**, with the only modification being the change in the anycast nodes. Instead of the initial configuration of $[3, 10]$, the anycast nodes are now set to $[5, 14]$.

Service	Worst round-trip Delay(ms)	Average round-trip Delay(ms)
1	11.98	6.24
2	12.00	6.15
3	4.90	2.52

Table 4.7: Table showing the WORST and AVERAGE round-trip delays in ms for each of the services provided. Service 1 and service 2 are UNICAST services, and for Servide 3 the path is routed through the shortest propagation delay path towards one of the choosen anycast nodes [5 14]

Total nº of cycles:	3394
Time of the best Solution(ms):	58.18
Cycle of Best solution(ms):	2
Best Objective(Gbps):	96.10
Average Objective(Gbps):	98.40

Table 4.8: Table containing the values for the Total number of cycles, Time of the best solution, The cycle where the best solution was found, the Best objective and the Average Objective for anycast nodes [5 14]

4.5 Exercise: 3e

Required: Compare the results obtained in all experiments of this Task 3 and of the previous Task 2 and draw all meaningful conclusions both concerning the differences between the solutions and the differences between the performance of the algorithm.

The results obtained in **Task 3** confirm the expected increase in the *Worst Link-Load* when compared to **Task 2**. This outcome is reasonable, as the *Hill Climbing Optimization* in Task 3 accounts for the additional constraints introduced by the **protection paths** required for service $S = 2$. Specifically, the link load for each flow in service $S = 2$ is the **sum of the working path load and the protection path reservation**. This added reservation for every flow increases the cumulative load on the network, resulting in higher worst link-load values across all configurations.

The path selection in Task 3 carefully balances these loads, minimizing the worst link-load while considering the added constraints of the protection paths. This optimization approach assures a balanced load distribution despite the increased demands on the network.

Best Node Configuration

From the analysis of all configurations, the **anycast nodes [5, 14]** consistently provided the

best performance in both Task 2 and Task 3, offering the lowest *Worst Link-Load* and competitive round-trip delays. The key results for **[5, 14]** are as follows:

Task 2 Results

- **Worst Link-Load**: 60.00 Gbps
- **Worst Round-Trip Delay** ($S = 3$): 4.90 ms
- **Average Round-Trip Delay** ($S = 3$): 2.52 ms

Task 3 Results

- **Worst Link-Load**: 96.10 Gbps
- **Worst Round-Trip Delay** ($S = 3$): 4.90 ms
- **Average Round-Trip Delay** ($S = 3$): 2.52 ms

The configuration **[5, 14]** strikes a balance between minimizing the worst link-load and maintaining acceptable delay metrics, even with the additional constraints of protection paths in Task 3.

Analysis of Anycast Nodes [3, 10]

In the configuration using **anycast nodes [3, 10]**, the *Worst Link-Load* exceeds the maximum link capacity of **100 Gbps**, which represents a critical issue. This occurs due to the suboptimal load distribution caused by the routing configuration, where specific links bear a disproportionately high load. The key results for **[3, 10]** in Task 3 are:

- **Worst Link Load**: 111.90 Gbps (exceeds capacity)
- **Worst Round-Trip Delay** ($S = 1$): 11.27 ms
- **Average Round-Trip Delay** ($S = 1$): 6.72 ms

This result highlights the importance of careful node selection and load balancing. Configurations like **[3, 10]** can result in network bottlenecks and potential link failures, emphasizing the need for optimization techniques that account for maximum link capacities during the routing process.

The comparison between Task 2 and Task 3 demonstrates the significant impact of introducing protection paths on network performance. While Task 2 achieves lower worst link-loads, Task 3

incorporates the necessary constraints for service reliability, resulting in higher but predictable link loads.

The **best-performing configuration** was achieved with **anycast nodes [5, 14]**, which provided a good balance between minimizing the *Worst Link-Load* and maintaining competitive *round-trip delays*. On the other hand, configurations like **[3, 10]** showed critical performance issues, exceeding the maximum link capacity and underlining the need for effective optimization and node selection strategies.

5. Information

The members' contributions were equal.
Our auto-evaluation is 17.