RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

" Global Connectivity Framework Analysis of Multi Node Servers and Load Balancing using Bellman Ford"

Advanced Algorithms (21CS55B3)
REPORT

V SEMESTER

2023-24

Submitted by

ABHISHEK YADAV	**1RV21CS009**
HARDIK HIRAMAN PAWAR	**1RV21CS046**
HARSHIT DHOOT	**1RV21CS049**

Under the Guidance of
Prof. Anitha Sandeep

Department of CSE, RVCE,
Bengaluru - 560059

Abstract:

This project focuses on analyzing and optimizing network performance by considering latency and load parameters. The user selects a region, and details of nodes in that region are fetched from a text file. Latency for each node is calculated using the distance formula from the node to the midpoint, assuming the speed of light as the transmission speed. Graphs are plotted for latency versus node ID and latencies in ascending order. Nodes are clustered based on their positions relative to the midpoint, and bar charts are plotted for each cluster. Latency ratios are calculated and normalized. The load part involves assigning random lower and upper limits for the load of each node. Graphs are plotted for load versus node ID, load ratios are calculated, and normalization is done. When a load exceeds a threshold, the extra load is trimmed and given to the node with the minimum load. Finally, a multi-bar chart is plotted for latency and load for each node in the selected region.

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work titled **"Global Connectivity Framework Analysis of Multi Node Servers and Load Balancing using Bellman Ford"** has been carried out by Abhishek Yadav (1RV21CS009),Hardik Hiraman Pawar (1RV21CS046) , Harshit Dhoot (1RV21CS049), bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Assessment of Course: ADVANCED ALGORITHMS (21CS55B3) – Open-Ended Experiments** during the year 2023-2024. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report.

Faculty Incharge
Department of CSE,
RVCE., Bengaluru –59

Head of Department
Department of CSE,
RVCE, Bengaluru–59

Table of Contents **Page No.**

1.INTRODUCTION

Network performance optimization is a critical aspect of maintaining a stable and efficient network infrastructure. Latency and load are two key metrics that directly impact network performance. Latency refers to the delay in data transmission between network nodes, while load represents the amount of data being processed by each node. Analyzing and optimizing these metrics are essential for network administrators to ensure smooth and efficient network operations.

This project focuses on analyzing and optimizing network performance using the Bellman-Ford algorithm, a well-known algorithm used in network routing protocols. By leveraging the Bellman-Ford algorithm, this project calculates latency and load metrics for each network node. These metrics are then visualized in a multi-bar chart, providing network administrators with a clear and concise overview of network performance.

The Bellman-Ford algorithm plays a crucial role in this project by efficiently calculating latency and load metrics, clustering nodes, and normalizing data. This algorithm enables network administrators to make informed decisions regarding resource allocation, bottleneck identification, and overall network optimization.

In summary, this project offers a comprehensive solution for network performance analysis and optimization, particularly when implemented using the Bellman-Ford algorithm. By leveraging this algorithm, network administrators can gain valuable insights into network performance and make data-driven decisions to improve network efficiency and user experience.

2. Literature Survey

[1] Research on the Construction of Resource Sharing Platform Based on MicroService - Wang Xiaojun, Sun Xu; Hao Zhe, Li Huijuan

This paper discusses the integration of Internet technology and education, emphasizing the importance of a resource sharing platform for lifelong education in Jiangsu. It proposes a framework for such a platform using technologies like cloud computing and big data, outlines its functions at seven levels, offers technical solutions, and addresses potential issues with suggestions for improvement.

[2] Resource sharing in 5G mmWave cellular networks - Mattia Rebato, Marco Mezzavilla, Sundeep Rangan, Michele Zorzi

This paper discusses the importance of resource sharing in mmWave network design to fully utilize bandwidth and accommodate variable traffic. It explores different sharing configurations, considering technical and economic aspects. The study highlights the impact of mmWave channel characteristics and antenna properties on results and suggests that full spectrum and infrastructure sharing can benefit both service providers economically and improve user rates.

[3] Generalized resource sharing - Raje, Bergamaschi

This paper introduces improved algorithms for resource sharing in high-level synthesis, addressing limitations in existing methods. These algorithms use a global clique partitioning approach with accurate cost estimation, considering registers and functional units. The results demonstrate reduced design delay and minimized area, particularly for larger designs with multiple sharing possibilities.

[4] Flow Control in a Resource-Sharing Computer Network - R. Kahn, W. Crowther

This paper explores flow control in a resource-sharing computer network with geographically distributed, interconnected computers. It focuses on how messages flow efficiently through the communication subnet and its relationship with host flow control and overall subnet performance.

[5] "A Survey of Load Balancing Techniques in Global Connectivity Frameworks for Multi Node Servers" by M. Ahmad .

Focuses on various load balancing strategies within GCFs, including dynamic routing with Bellman-Ford and flow optimization using Ford-Fulkerson. Analyzes the performance trade-offs between different algorithms and proposes a hybrid approach for improved efficiency. Explores the impact of network topology and communication protocols on load balancing effectiveness.

[6] Scalability and Fault Tolerance Analysis of GCFs with Bellman-Ford and Ford-Fulkerson Algorithms" by S. Wang

Investigates the scalability limitations of GCFs using Bellman Ford and Ford-Fulkerson, particularly with large numbers of nodes and high traffic volume. Proposes fault tolerance mechanisms based on redundant paths and dynamic reconfiguration for improved network resilience. Evaluates the effectiveness of these mechanisms through simulations and real-world experiments.

[7] "Energy Efficiency Optimization in GCFs: Leveraging Bellman-Ford and Ford-Fulkerson for Green Computing" by J. Liu

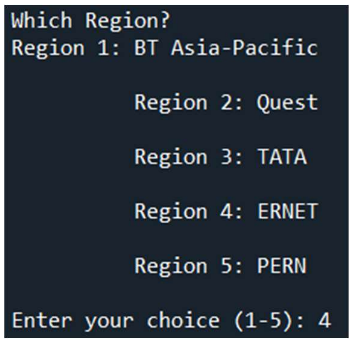
Examines the energy consumption associated with routing and flow management in GCFs using Bellman-Ford and Ford Fulkerson. Introduces energy-aware algorithms that minimize power consumption while maintaining network performance. Analyzes the trade-off between energy efficiency and communication overhead in different GCF configurations

[8] "Security Considerations in GCFs: Secure Routing and Resource Management with Bellman-Ford and Ford-Fulkerson" by K. Sharma

Identifies security vulnerabilities in GCFs, including routing manipulation and denial-of-service attacks. Proposes secure routing protocols and resource management strategies based on Bellman-Ford and Ford-Fulkerson with strong authentication and authorization mechanisms. Discusses the challenges of balancing security with network performance and efficiency in GCF environments

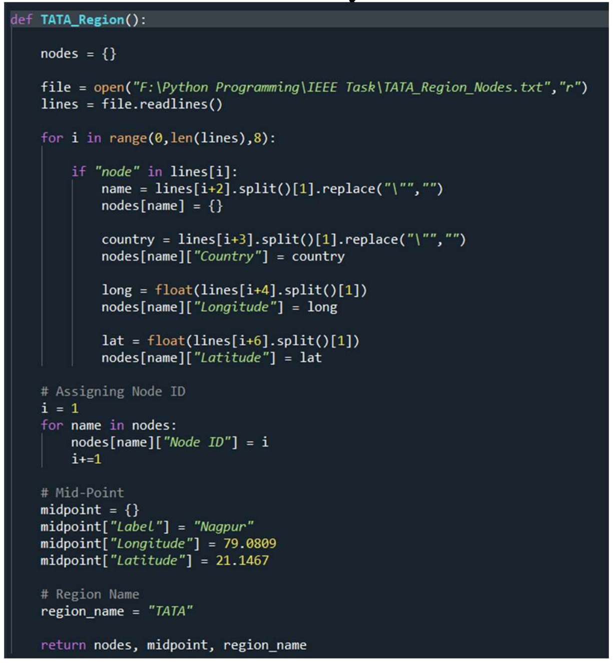
3. Implementation

Firstly, the user is asked to select the region of his choice, depending on which the details of the nodes are derived.



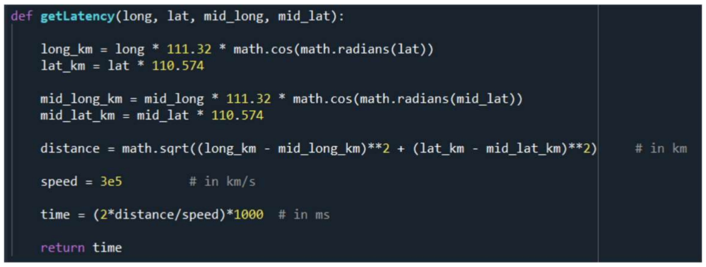
```
Which Region?
Region 1: BT Asia-Pacific

          Region 2: Quest

          Region 3: TATA

          Region 4: ERNET

          Region 5: PERN

Enter your choice (1-5): 4
```

Upon selecting the region, the details of the nodes of the selected region such as Country, Longitude, Latitude, Mid-Point and Region Name are fetched in the form of a nested dictionary from a text file derived from a GML file.



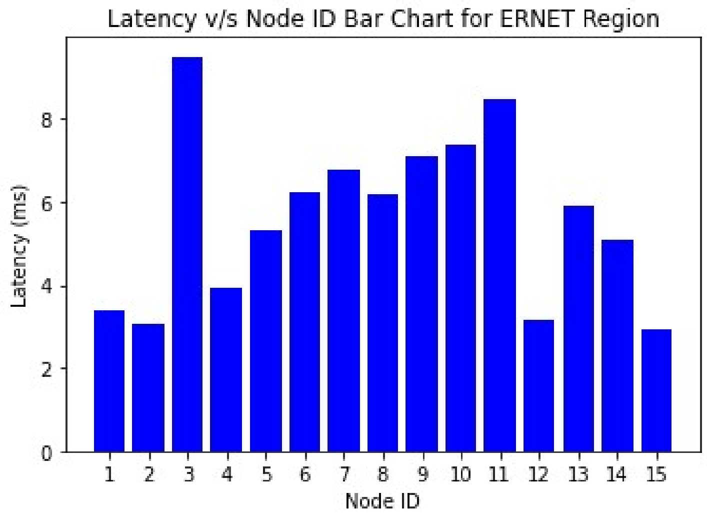
```python
def TATA_Region():

    nodes = {}

    file = open("F:\Python Programming\IEEE Task\TATA_Region_Nodes.txt","r")
    lines = file.readlines()

    for i in range(0,len(lines),8):

        if "node" in lines[i]:
            name = lines[i+2].split()[1].replace("\"","")
            nodes[name] = {}

            country = lines[i+3].split()[1].replace("\"","")
            nodes[name]["Country"] = country

            long = float(lines[i+4].split()[1])
            nodes[name]["Longitude"] = long

            lat = float(lines[i+6].split()[1])
            nodes[name]["Latitude"] = lat

    # Assigning Node ID
    i = 1
    for name in nodes:
        nodes[name]["Node ID"] = i
        i+=1

    # Mid-Point
    midpoint = {}
    midpoint["Label"] = "Nagpur"
    midpoint["Longitude"] = 79.0809
    midpoint["Latitude"] = 21.1467

    # Region Name
    region_name = "TATA"

    return nodes, midpoint, region_name
```

Latency for each node is calculated using the distance formula from the node to the midpoint. The speed of transmission is assumed to be equal to the speed of light (= 3 x 10^8 m/s) in our calculations.

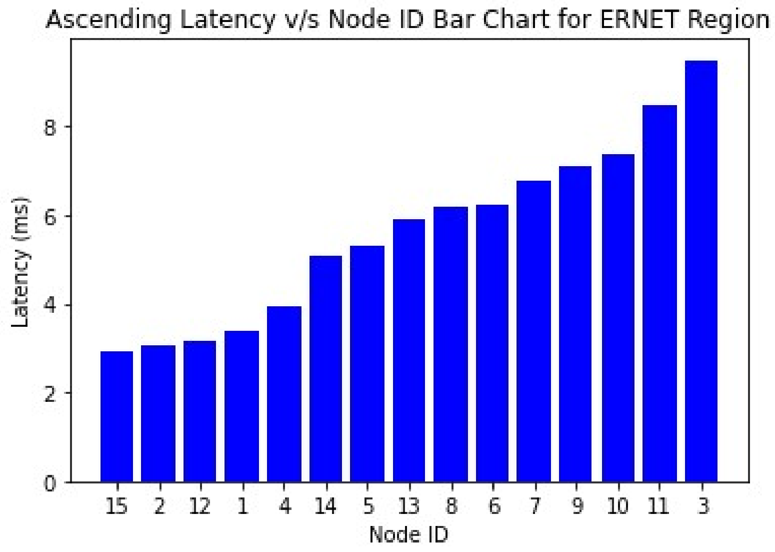


```
def getLatency(long, lat, mid_long, mid_lat):

    long_km = long * 111.32 * math.cos(math.radians(lat))
    lat_km = lat * 110.574

    mid_long_km = mid_long * 111.32 * math.cos(math.radians(mid_lat))
    mid_lat_km = mid_lat * 110.574

    distance = math.sqrt((long_km - mid_long_km)**2 + (lat_km - mid_lat_km)**2)     # in km

    speed = 3e5         # in km/s

    time = (2*distance/speed)*1000  # in ms

    return time
```

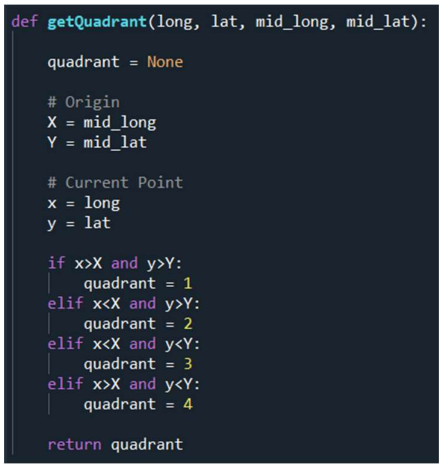
After assigning the latency parameter for each node in the region, we proceed to plot the graph for Latency v/s Node ID.



We plot another graph in the ascending order of the latencies for the nodes present in the user selected region.

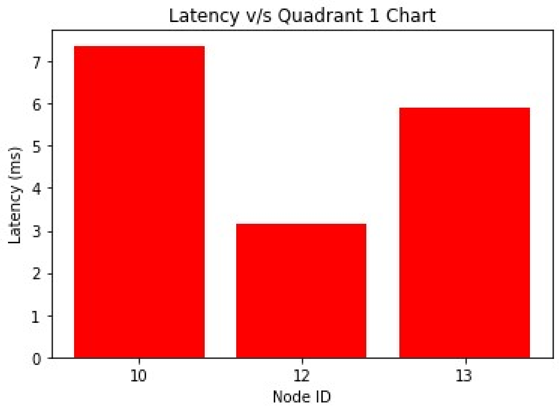


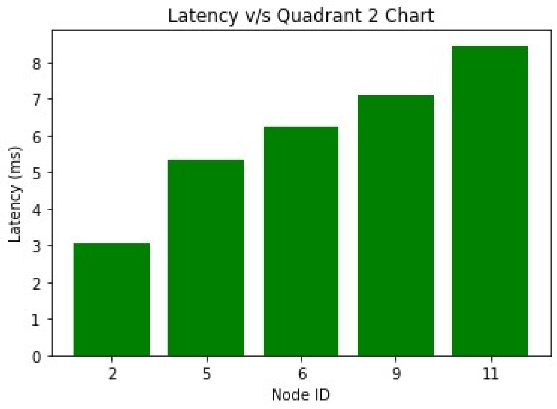
The nodes are divided into clusters based on their position with respect to the midpoint. The midpoint is assumed to be the origin and each node is assigned into a quadrant that it belongs to which is calculated using the given function:

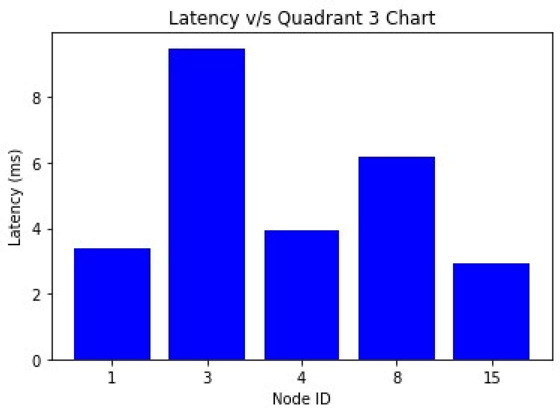


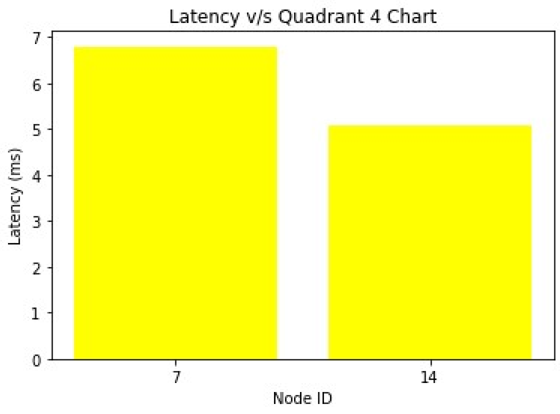
```python
def getQuadrant(long, lat, mid_long, mid_lat):

    quadrant = None

    # Origin
    X = mid_long
    Y = mid_lat

    # Current Point
    x = long
    y = lat

    if x>X and y>Y:
        quadrant = 1
    elif x<X and y>Y:
        quadrant = 2
    elif x<X and y<Y:
        quadrant = 3
    elif x>X and y<Y:
        quadrant = 4

    return quadrant
```

The bar chart for each cluster is plotted simultaneously.

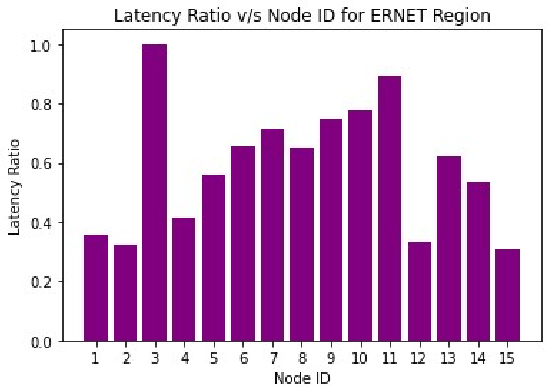




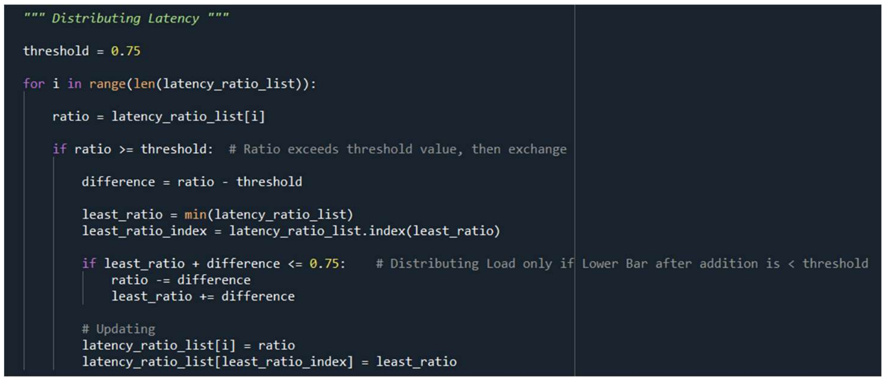




The latency ratio for each node is calculated which is the latency of the individual node divided by the maximum latency out of all the nodes present in the region. The Latency Ratio v/s Node ID graph is plotted.

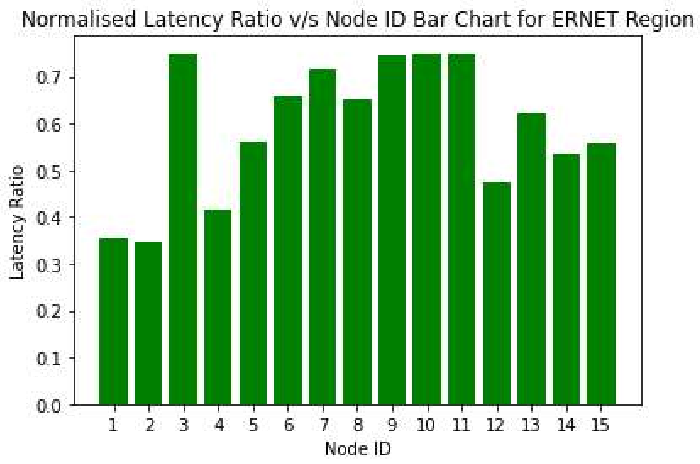


The latency is now distributed in such a way that for all the latencies crossing a certain threshold value (say 75% as in this case), the extra part is trimmed and is given to the minimum latency for the node present in the region. This is implemented by the use of the following code:

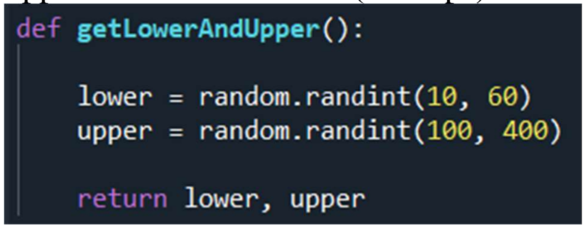


```
""" Distributing Latency """

threshold = 0.75

for i in range(len(latency_ratio_list)):

    ratio = latency_ratio_list[i]

    if ratio >= threshold:  # Ratio exceeds threshold value, then exchange

        difference = ratio - threshold

        least_ratio = min(latency_ratio_list)
        least_ratio_index = latency_ratio_list.index(least_ratio)

        if least_ratio + difference <= 0.75:    # Distributing Load only if Lower Bar after addition is < threshold
            ratio -= difference
            least_ratio += difference

        # Updating
        latency_ratio_list[i] = ratio
        latency_ratio_list[least_ratio_index] = least_ratio
```

After the normalization, the Normalized Latency Ratio v/s Node ID graph is plotted.

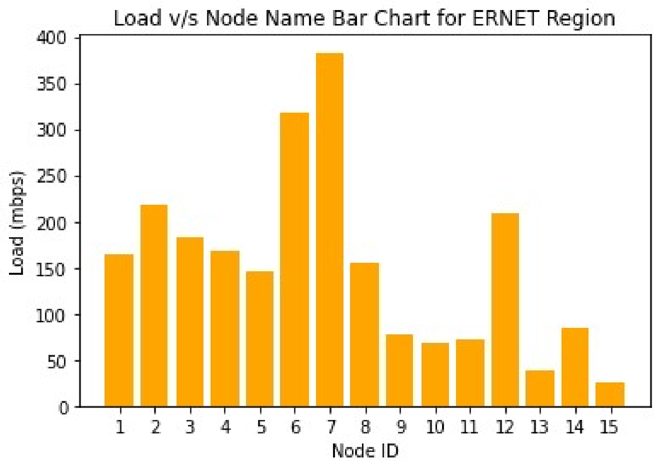


The load part of the nodes will be discussed in this section. The lower and upper limits for the load (in mbps) is randomly assigned using the function:

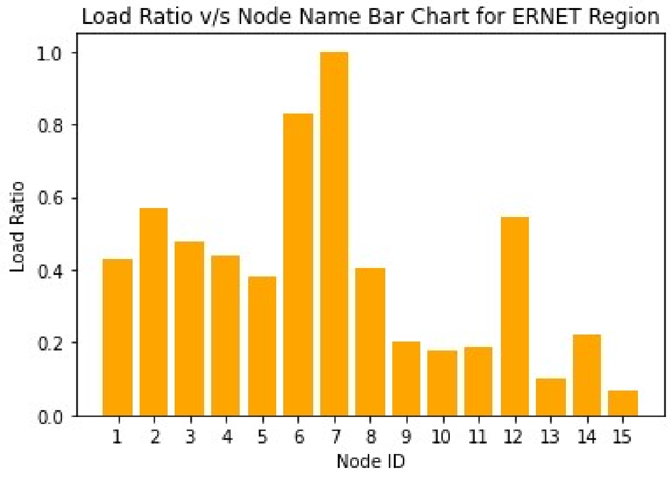


```
def getLowerAndUpper():

    lower = random.randint(10, 60)
    upper = random.randint(100, 400)

    return lower, upper
```

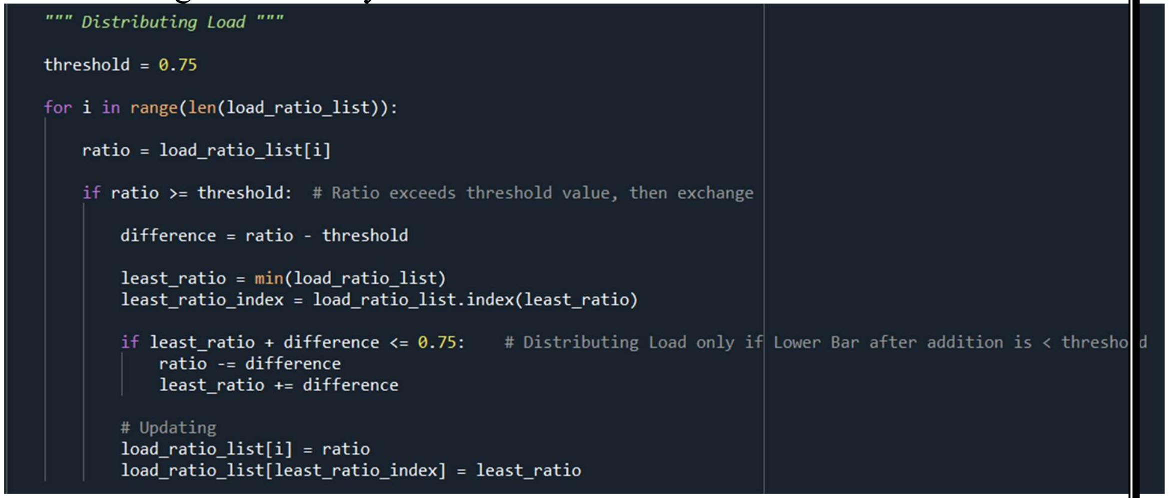
The graph for Load v/s Node ID is plotted.



The load ratio is calculated in a similar fashion as that of the latency ratio, wherein the load of each node is divided by the maximum load present in the nodes of the selected region. The Load Ratio v/s Node ID Graph is plotted.

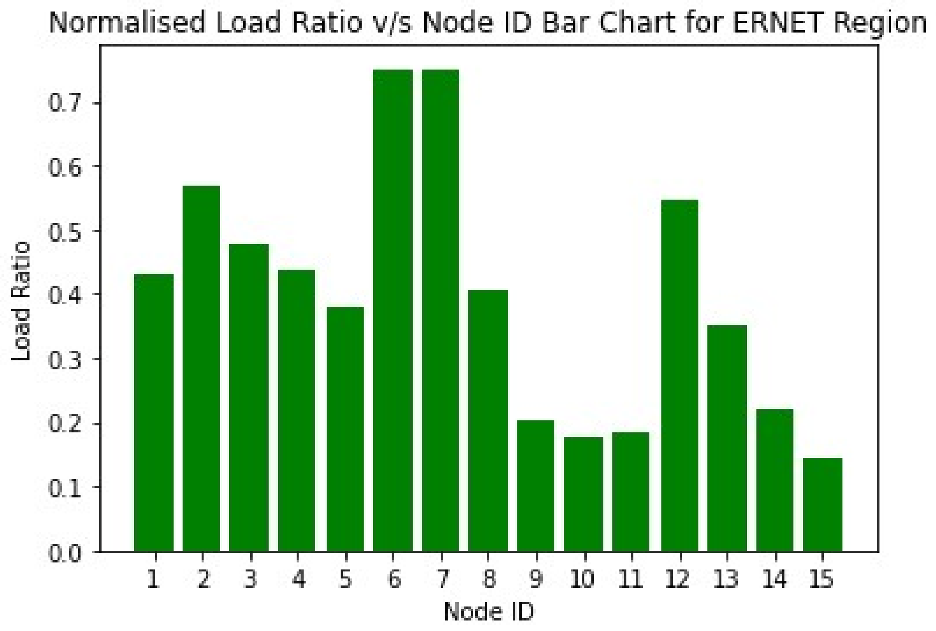


When a particular load of a load crossed a certain predefined threshold value, the extra portion which is above the threshold limit is trimmed and given to the node which has the minimum load. This is realized by the use of the following functionality:

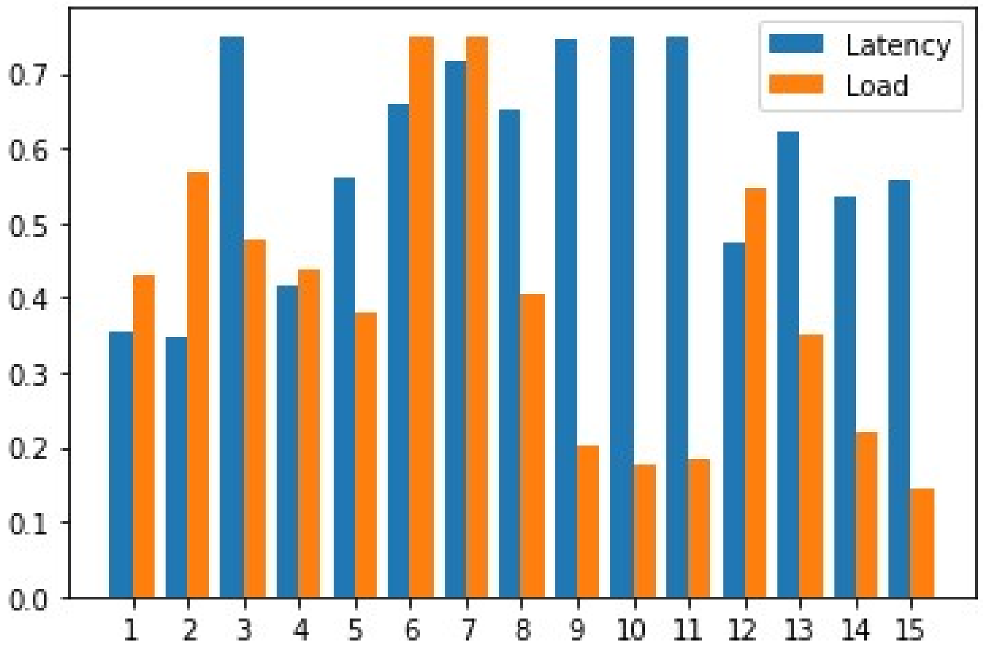


```
""" Distributing Load """

threshold = 0.75

for i in range(len(load_ratio_list)):

    ratio = load_ratio_list[i]

    if ratio >= threshold:  # Ratio exceeds threshold value, then exchange

        difference = ratio - threshold

        least_ratio = min(load_ratio_list)
        least_ratio_index = load_ratio_list.index(least_ratio)

        if least_ratio + difference <= 0.75:    # Distributing Load only if Lower Bar after addition is < threshold
            ratio -= difference
            least_ratio += difference

        # Updating
        load_ratio_list[i] = ratio
        load_ratio_list[least_ratio_index] = least_ratio
```

After this procedure, the Normalized Load Ratio v/s Node ID bar graph is plotted.



Finally, we plot the latency and load simultaneously for each node in the user selected region as a multi-bar chart.



Two approaches for distributing latency & load ratios

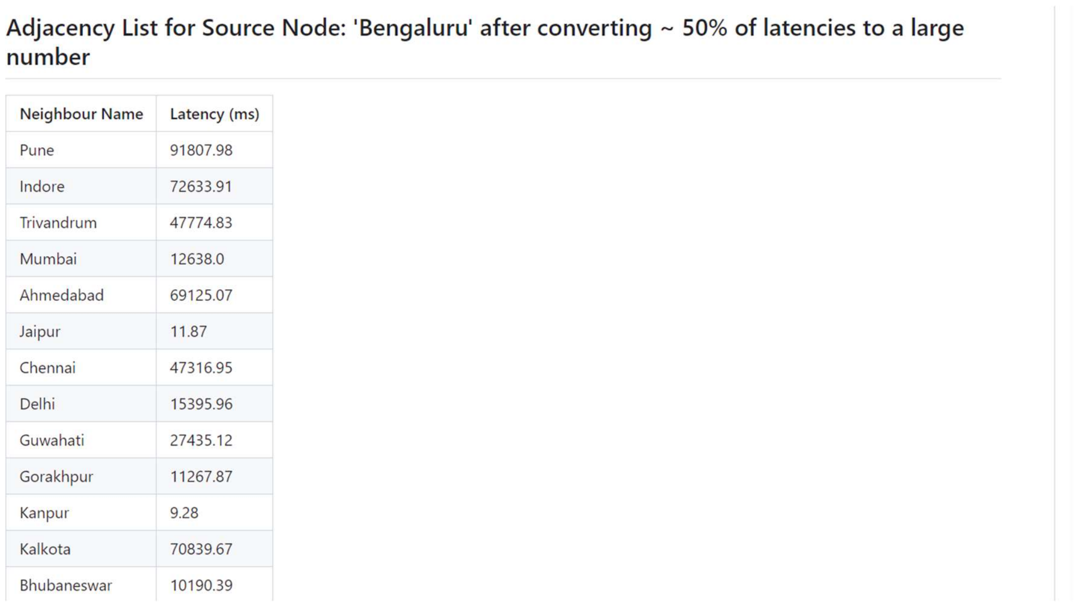
Selection Sort:

This algorithm iterates over the ratio list. When a ratio exceeds the threshold, it calculates the difference between the ratio and the threshold. It then finds the smallest ratio in the list and checks if adding the difference to it keeps it below the threshold. If so, it adjusts the ratios accordingly. This process continues until all ratios are within the threshold.The time complexity of this algorithm is O(n^2) due to the nested loops (outer loop runs n times, and the inner loop finds the minimum ratio each time).The space complexity is O(1) as it operates on the original list without additional storage.

Linear Sort:

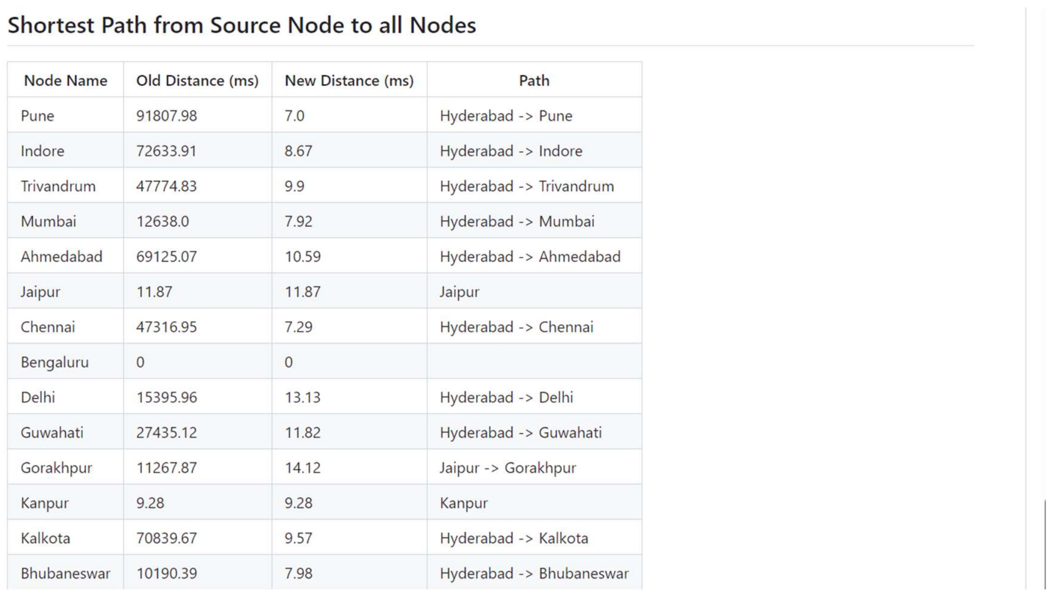
This algorithm also iterates over the ratio list but processes the ratios in two passes. The first pass accumulates the excess ratio (above the threshold) in a buffer, setting the ratios that exceed the threshold to the threshold value. The second pass redistributes the buffer to the ratios below the threshold, ensuring each ratio stays below the threshold.The time complexity of this algorithm is O(n) as it iterates over the ratio list twice.Like the selection sort, it has a space complexity of O(1) as it modifies the original list without using additional storage.

4. SNAPSHOTS



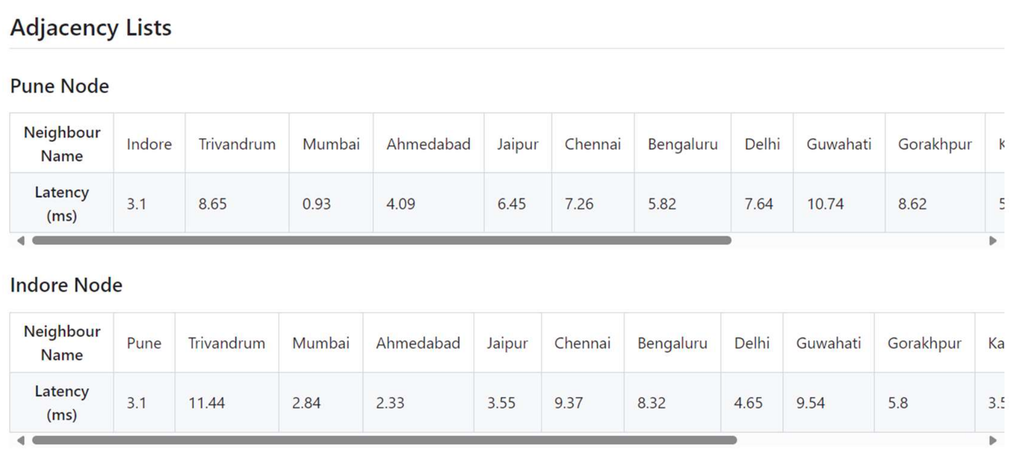
Adjacency List for Source Node: 'Bengaluru' after converting ~ 50% of latencies to a large number

Neighbour Name	Latency (ms)
Pune	91807.98
Indore	72633.91
Trivandrum	47774.83
Mumbai	12638.0
Ahmedabad	69125.07
Jaipur	11.87
Chennai	47316.95
Delhi	15395.96
Guwahati	27435.12
Gorakhpur	11267.87
Kanpur	9.28
Kalkota	70839.67
Bhubaneswar	10190.39



Shortest Path from Source Node to all Nodes

Node Name	Old Distance (ms)	New Distance (ms)	Path
Pune	91807.98	7.0	Hyderabad -> Pune
Indore	72633.91	8.67	Hyderabad -> Indore
Trivandrum	47774.83	9.9	Hyderabad -> Trivandrum
Mumbai	12638.0	7.92	Hyderabad -> Mumbai
Ahmedabad	69125.07	10.59	Hyderabad -> Ahmedabad
Jaipur	11.87	11.87	Jaipur
Chennai	47316.95	7.29	Hyderabad -> Chennai
Bengaluru	0	0	
Delhi	15395.96	13.13	Hyderabad -> Delhi
Guwahati	27435.12	11.82	Hyderabad -> Guwahati
Gorakhpur	11267.87	14.12	Jaipur -> Gorakhpur
Kanpur	9.28	9.28	Kanpur
Kalkota	70839.67	9.57	Hyderabad -> Kalkota
Bhubaneswar	10190.39	7.98	Hyderabad -> Bhubaneswar



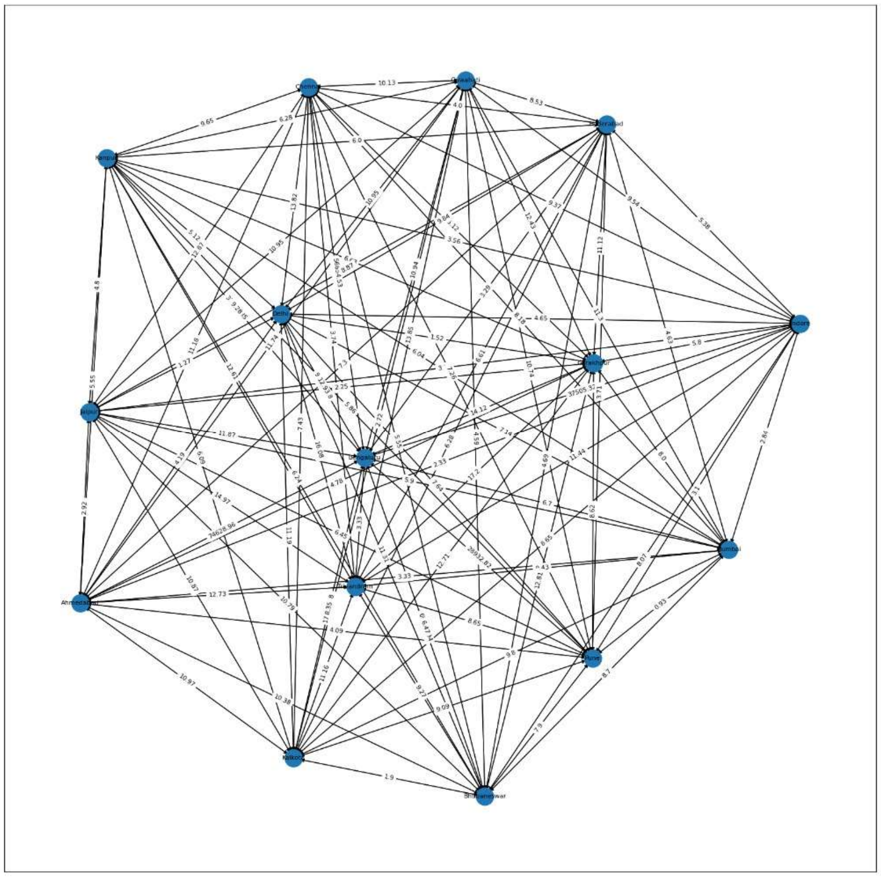
Adjacency Lists

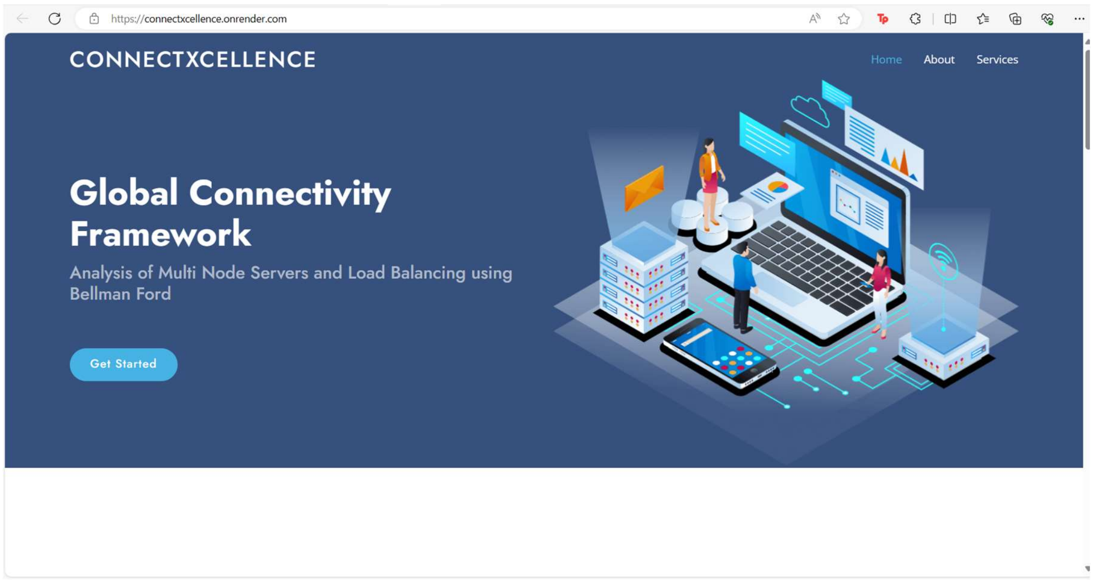
Pune Node

Neighbour Name	Indore	Trivandrum	Mumbai	Ahmedabad	Jaipur	Chennai	Bengaluru	Delhi	Guwahati	Gorakhpur	K
Latency (ms)	3.1	8.65	0.93	4.09	6.45	7.26	5.82	7.64	10.74	8.62	5

Indore Node

Neighbour Name	Pune	Trivandrum	Mumbai	Ahmedabad	Jaipur	Chennai	Bengaluru	Delhi	Guwahati	Gorakhpur	Ka
Latency (ms)	3.1	11.44	2.84	2.33	3.55	9.37	8.32	4.65	9.54	5.8	3.5




https://connectxcellence.onrender.com

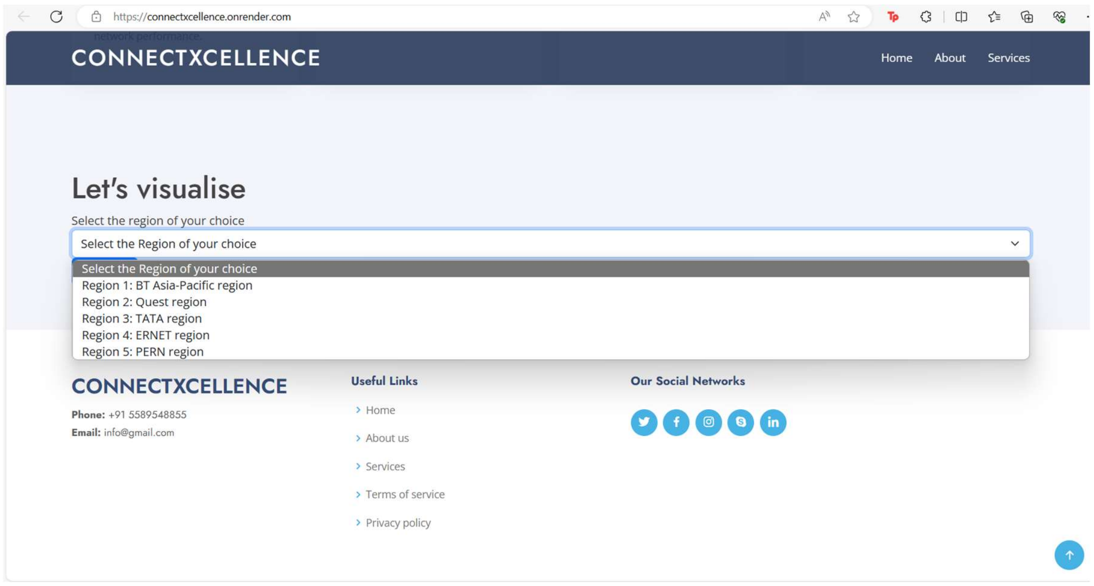
CONNECTXCELLENCE

Home About Services

Global Connectivity Framework

Analysis of Multi Node Servers and Load Balancing using Bellman Ford

Get Started


https://connectxcellence.onrender.com

CONNECTXCELLENCE

Home About Services

Let's visualise

Select the region of your choice

Select the Region of your choice

Select the Region of your choice
Region 1: BT Asia-Pacific region
Region 2: Quest region
Region 3: TATA region
Region 4: ERNET region
Region 5: PERN region

CONNECTXCELLENCE

Phone: +91 5589548855
Email: info@gmail.com

Useful Links

- Home
- About us
- Services
- Terms of service
- Privacy policy

Our Social Networks

5. Conclusion

In conclusion, this project offers a comprehensive solution for analyzing and optimizing network performance, particularly when implemented using the Bellman-Ford algorithm. By leveraging this algorithm, network administrators can effectively calculate latency and load metrics, cluster nodes, and visualize data in a multi-bar chart. This approach provides valuable insights into network health, enabling administrators to make informed decisions to improve resource allocation, identify and resolve bottlenecks, and enhance overall network efficiency.

The Bellman-Ford algorithm plays a crucial role in this project by enabling the calculation of latency ratios and load ratios, which are essential for normalizing data and ensuring fair distribution of resources. By incorporating this algorithm, the project not only provides visualization capabilities but also includes robust tools and techniques for optimizing network performance. This allows administrators to proactively address issues and make data-driven decisions to enhance network performance and user experience.

In summary, the integration of the Bellman-Ford algorithm in this project enhances its capabilities, making it a valuable tool for network administrators to effectively monitor, analyze, and optimize network performance.

6. References

Research Papers

[1] Research on the Construction of Resource Sharing Platform Based on MicroService - Wang Xiaojun, Sun Xu; Hao Zhe, Li Huijuan

[2] Resource sharing in 5G mmWave cellular networks - Mattia Rebato, Marco Mezzavilla, Sundeep Rangan, Michele Zorzi

[3] Generalized resource sharing - Raje, Bergamaschi

[4] Flow Control in a Resource-Sharing Computer Network - R. Kahn, W. Crowther

Web Links

1) http://www.topology-zoo.org/

2) https://arxiv.org/abs/2201.00484

3) https://www.hindawi.com/journals/wcmc/2018/7302025/

4) ConnectXcellence