**MPPLAB – Graph Representation of TelCom Traffic for MPLS Based BSNL Network – Telcom Math Model\***

(**\*** = Part of **MPPLab Project)**

**Report No. 1**

**Draft Version - 1.0**

**May 20, 2025**

**Authors**

VVIT Students (2023 batch) Internship Students & MPPLab Project Staff

This page is intentionally left blank.

# Table of Contents

1. **Part-I: Overview..............................................................................................**

5

5

5

6

6

7

13

8

26

28

29

* 1. Objective.......................................................................................................
  2. Details of Project…………………………………………………………
     1. MPPLab and Overview of Telecom Math Model ………….…….
  3. Telecom Networks: Graph based Physical and Logical Topology .
  4. Input File Data for Graph Based BSNL Logical Topology ………..
  5. Programming Environment Used ……………………………………….
  6. Implementation – Construction of Graph based Logical Topology…..
  7. Description of Various Modules ……………………………………
  8. Results and Summary ………………………………………….……
  9. Limitations …………………………………………………….……..

This page is intentionally left blank.

1. **Part-I : Preprocessing Step** 
   1. **Objective:**

The objective(s) are to construct logical topology of typical Tele-traffic Network Data of MPPLAB Project such as BSNL Network based on graph theory; and integrate with MPPLAB Telecom Math model to generate necessary graph based input data for Multi-Commodity Flow Solver.

* 1. **Details of the Project** 
     1. **Background : MPPLAB and an overview of Telecom Math Model**

"MPPLab" stands for Mathematical Programming (in) Parallel Laboratory. One of the Objectives is to create a "Unified Architecture (UA)" Supercomputer to support mathematically oriented High Productivity Computing and its multiple applications. UA is the pre-requisite for all applications and consists of tools to undertake traffic engineering analysis in India.

UA purpose is to provide an HPC platform for real-time control applications of economic significance and facilitate efficient utilization of resources in large systems. CDAC, CDoT, and IISc create a "Unified Architecture (UA)" Supercomputer ( to support mathematically oriented High Productivity Computing platform for real-time control applications of economic significance such as teletraffic Engineering application. Another purpose is to provide an HPC platform for real-time control applications of economic significance and facilitate efficient utilization of resources in large systems.

The architecture diagram is shown in above figure 1, and it is special designed for runtime support to process the huge amount of data of teletraffic Engineering application. The term teletraffic covers all kinds of data communication traffic and telecommunication traffic,

In Telecommunication Applications, network traffic needs to be analyzed, improved and routed in real time. To couple the application algorithm in the core system to telecommunication system, it is necessary to adapt routers so that they can extract required data from telecom network to be given as input to parallel algorithm, and apply the output of the algorithm to the telecom network.

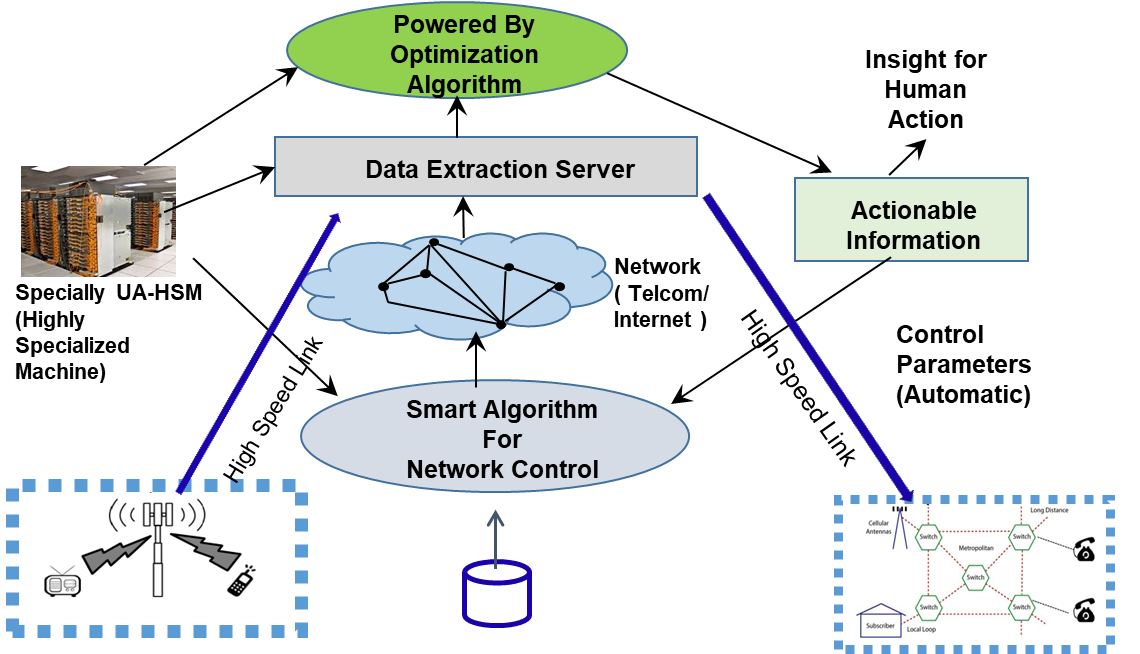


Figure 1 MPPLAB Architecture Diagram involving Tele-traffic Engineering Application

* 1. **MPPLAB and an overview of Telecom Math Model**

The Telecom Traffic Model is a simulation framework designed to estimate the volume of telecom traffic, such as phone calls, between different cities or localities. The model treats the telecommunication network as a **graph**, where each **city is a node**, and telecom links are the **edges**. It uses multiple real-world factors—such as **population, activity levels, distance, time zone, and language affinity**—to determine the communication intensity between city pairs The model consists of four key modules:

1. **Map Module** – Reads and stores basic city data like population, coordinates (latitude, longitude), and language. It also calculates the distance between cities and allows new city entries.
2. **Time Zone Module** – Adjusts time intervals according to a city’s longitudinal region. India is divided into five zones based on longitude. This ensures that activity levels align with local time.
3. **Activity Module** – Stores activity levels for 48 half-hour intervals over a day. Each city’s activity at a specific time is accessed after shifting time according to its time zone.
4. **Affinity Module** – Calculates the affinity between two cities based on their physical distance and language similarity. Cities that are closer and share the same language have higher affinity.

This value represents how busy the telecom traffic is expected to be between two cities at a given time Overall, the model acts like a **telecom forecasting engine**, helping to simulate usage patterns and plan capacity by combining demographic, geographic, temporal, and behavioral data. It is flexible, scalable, and allows for adding new data or refining calculations as needed

This is a mathematical and simulation model for telecom traffic behavior, especially focusing on user activity, geographic considerations, network affinity, codec management, and transitions to VoIP (Voice over IP) communication. This modular structure allows precise modeling of how telecom resources are allocated and consumed based on time, geography, and technology. This involves conversion of “number of calls” (output from Telecom Math Model v1.0) to VoIP packets and bytes based on the CoDec specifications.

The conversion is necessary because, in modern telecom networks, traditional voice calls are increasingly being replaced by **VoIP (Voice over Internet Protocol)** packets. Most modern communication networks (including 4G/5G, fiber, broadband) are built on **IP-based infrastructure**. To transmit calls over these networks, voice data must be **converted into IP packets**. After estimating the **number of calls** between cities, converting them into **VoIP packets** allows:

* Simulation of **packet-level traffic**
* Estimating **network load**
* Planning **packet routing, bandwidth allocation**, and **packet loss handling**

The additional modules added to the Telecom Math Model in version 2.0 are “codec\_module” and “calls\_to\_voip\_module”.

**Codec module:** This module defines the object oriented approach in Fortran for handling the codec specifications and computations of voice payload. It ensures that the VoIP packet size is taken on the basis of codec specifications like bitrate and packetization interval.

**Callisto module:** This module is the actual implementation for converting the number of phone calls to VoIP packets and computation of bandwidth.

Together, these modules form an interconnected framework for simulating telecom traffic with geographical, temporal, behavioral, and technological realism

**1.4 Telecom Networks: Graph based Physical and Logical Topology**

core segments of the network are usually implemented using two separate layers: an optical layer and a transport layer taking care of routing at the IP level. Multi-layer networks usually transport huge amount of data and a core network is composed of high-performance devices, each of them consuming tens power and energy and connected with optical links covering long-haul distances. Also, these networks employ several amplifiers which are needed to connect two endpoints of a single ink.

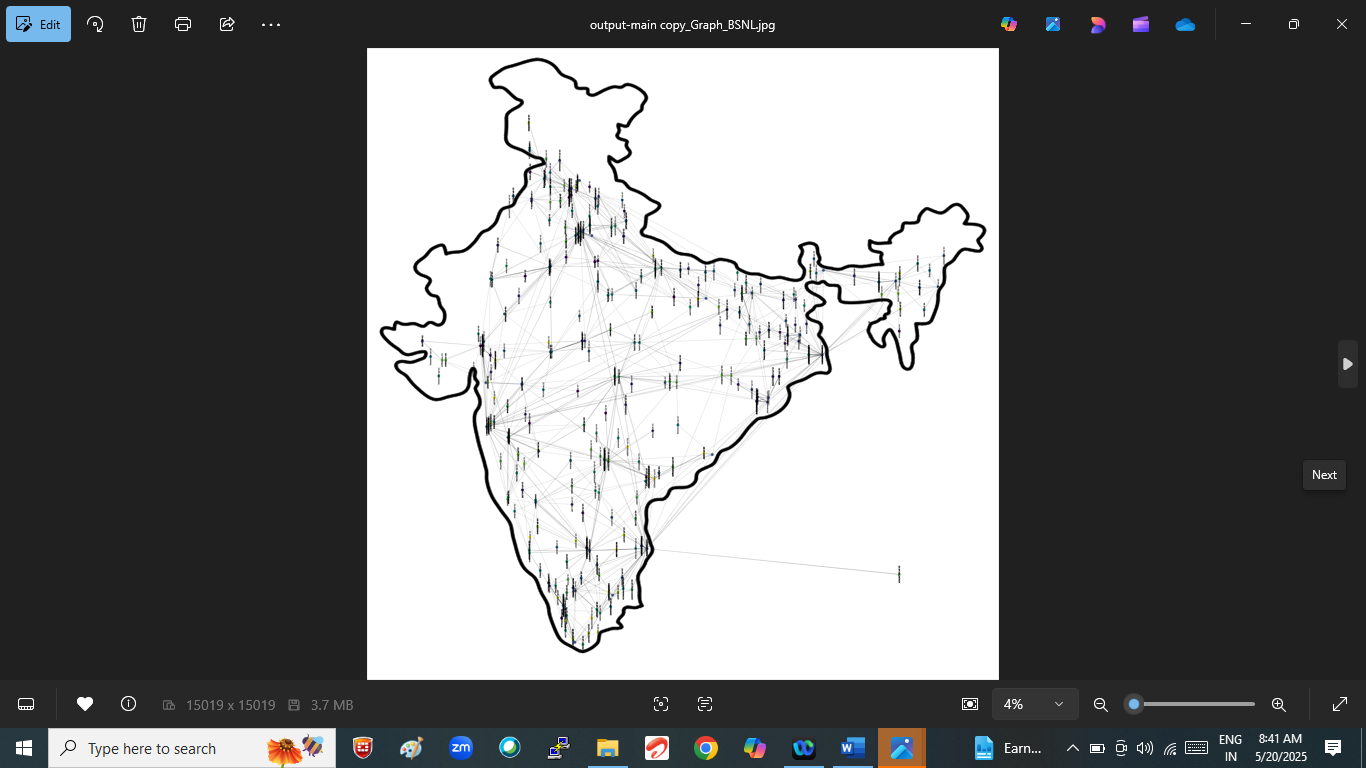
For convenience’s sake, we consider the entire network design of an IP-over- traditional network, meaning that we look not only at the IP layer deciding which routers and line cards to install and how to route the traffic over the set of IP links. In particular, all of them assume Multi-Commodity Flow (MCF), i.e., the typical transportation problem where multiple commodities (traffic demands) need to be routed over a network with limited capacity. However, MCF assumes that a traffic demand can be split over different paths. This assumption often cannot be applied in a realistic telecommunication network, since many routing protocols at the IP layer are constrained to Single- Path Routing (SPR), i.e. the traffic demand between a **source** and a **destination** is entirely routed over the same IP path. SPR and Multi-Path Routing (MPR) , both have their own benefits in terms of power consumption. In our experiments, we define two networks: **Physical Topology** and **Logical Topology.**

**Physical Topology (Discrete Data of BSNL Network)**

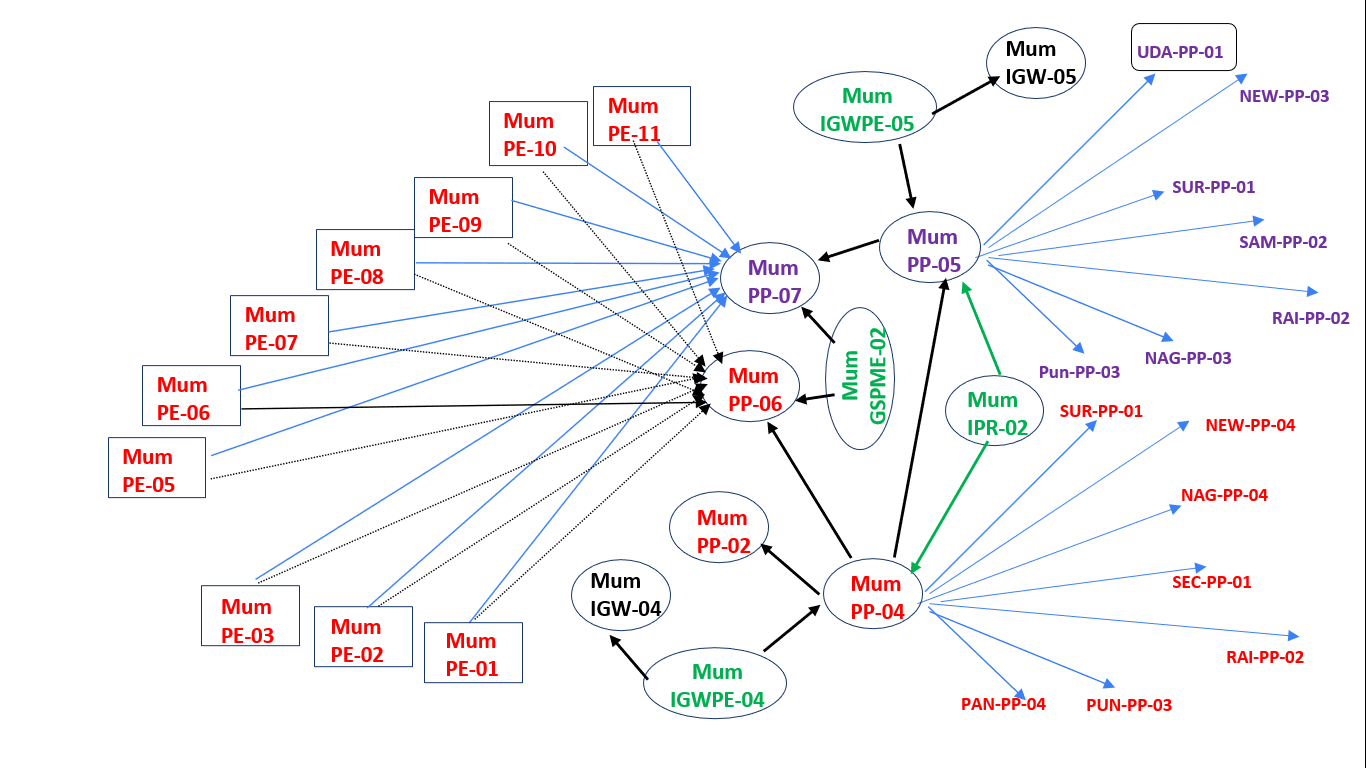
BSNL’s end-to-end transport network architecture consists of core routers, and Edge routers. Core Layer comprises of more than **100 Backbone Routers** as a part of Test Bed. All Core routers are interconnected over multiple bundles or physical **10G/100G** links. All edge locations have **two or more** core router(s) which are connected to core using **1G/10G or more** as connectivity. The discrete data consists of total number of different routers deployed and each one connectivity with others with characteristics as given below.

* Router1
* Router2
* #100G
* #10G
* #STM16
* GE
* Total BW
* Type
* PeakIN
* PeakOUT
* Region

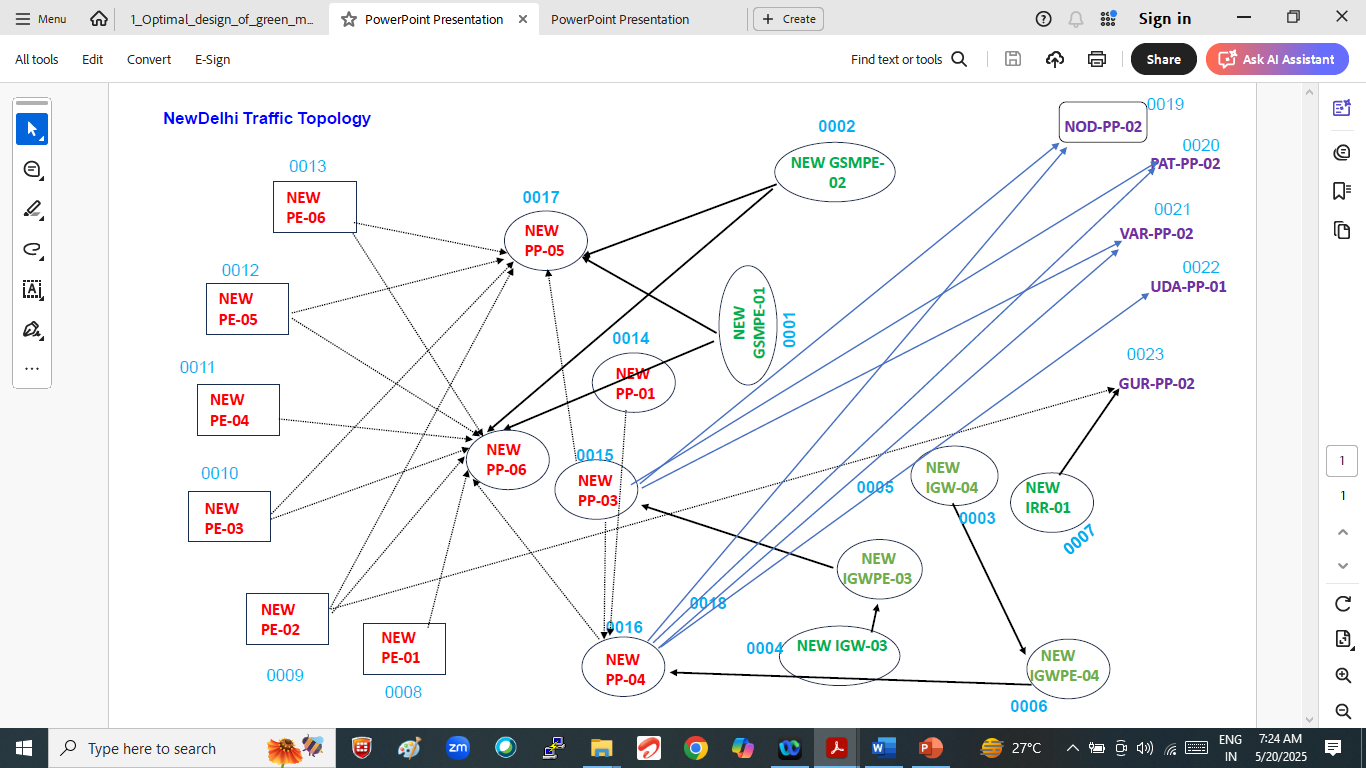
The graph representation of physical topology is shown in the Figure 2.



**Fig. 3(a). Physical topology of BSNL Network Complete Network**



**Fig. 3(b). Physical topology** of BSNL Network - Mumbai City – part of Complete Network

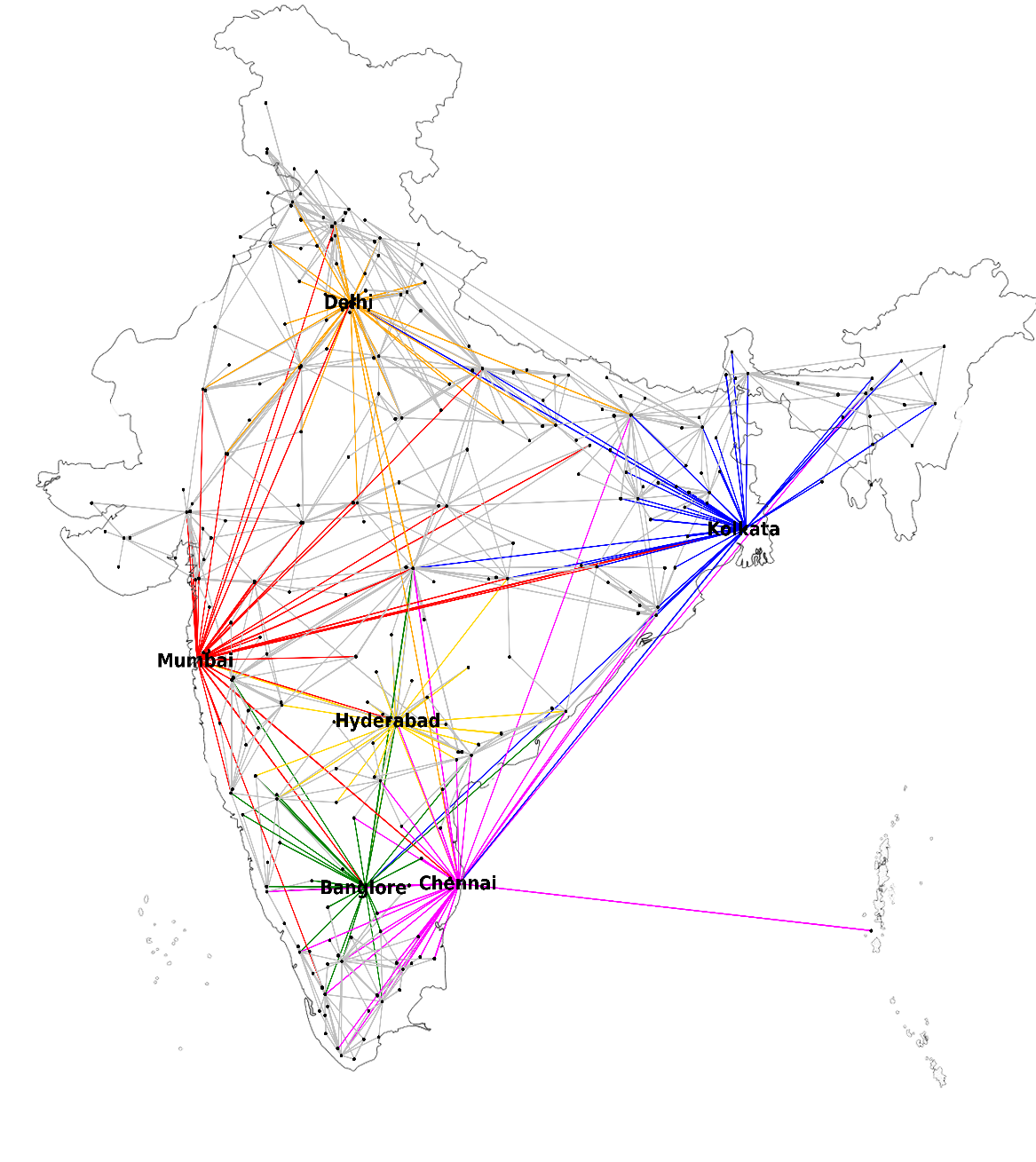


**Fig.3(c).** **Physical topology** of BSNL Network – New Delhi City – part of Complete Network

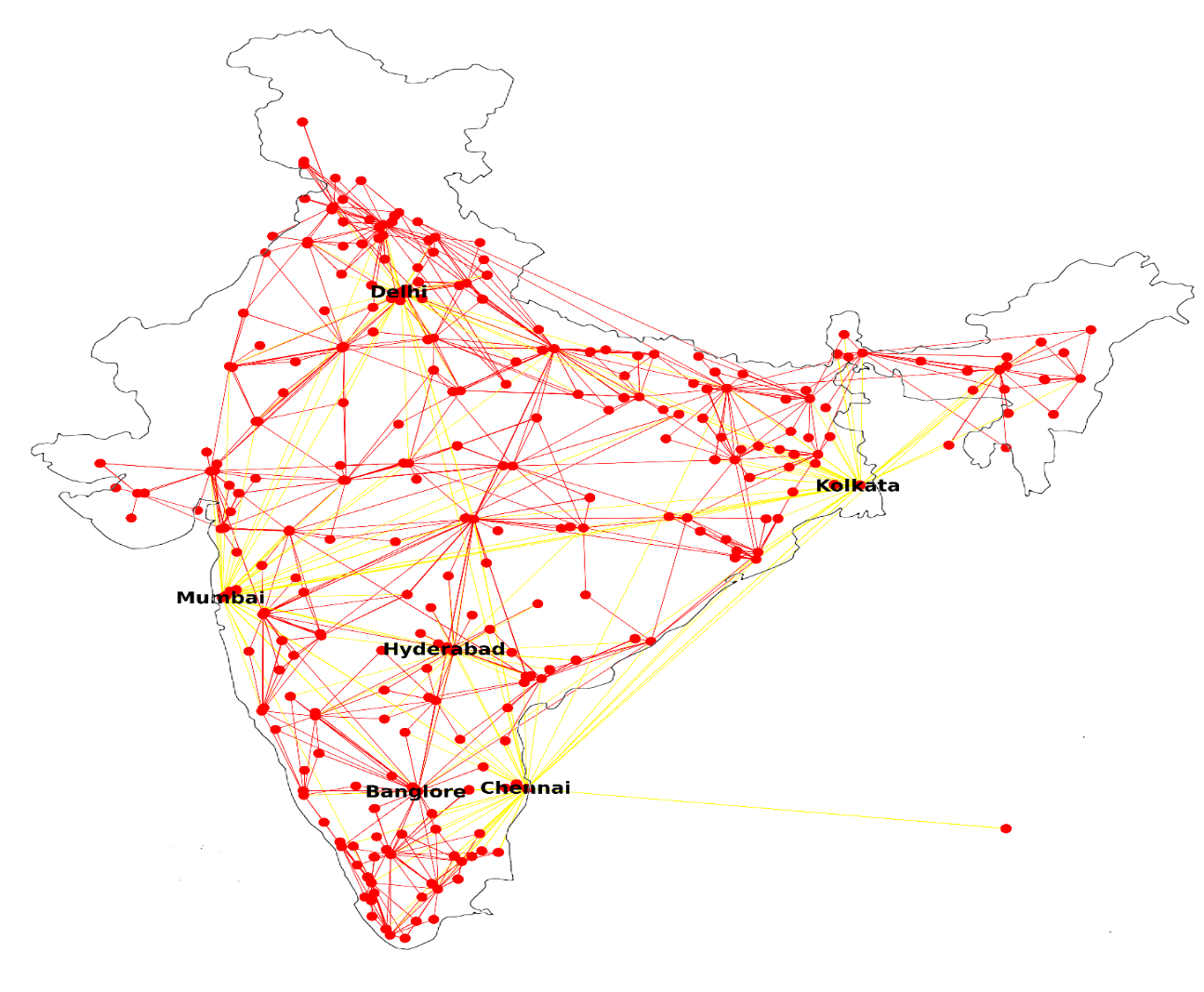
In Multi-layered Network Topology (BSNL), layered approach is followed to connect Tier-1, and Tie-2 Cities. Total Number of Routers: \*\*\*\* Total No. of Edges : \*\*\*\*

**Logical Topology (Discrete Data of BSNL Network)**

let us represent the physical supply network as an undirected graph **G = (V,E)** where **V** is the set of nodes (City Location) where routers are installed and **E** is the set of admissible physical links. Each node **i ∈ V** can be equipped with an IP router n out of the set **N** of IP routers. We introduce the set of **commodities K** based on point-to-point demands **dij**, **(i, j) ∈ V × V,** i **<** j. The set **K ⊆ V** corresponds to those nodes in **V** that are source of at least one demand. The Un-directed graph is shown in Figure 3(a) to Figure 3(b).

****

**Fig. 4(a).** Logical l topology of BSNL Network

****

**Fig. 4(b). Logical l topology of BSNL Network**

**1.5.Programming Environment Used**

* Multi-Core System with Ubuntu OS, Jupyter Notebook, Python, Networkx, Pandas, Metplotlib,Numpy,Sklearn

**1.6.Implementation – Construction of Graph based Logical Topology**

The project visualizes and analyzes telecom network traffic using city-wise router connection data. It processes a CSV file containing source and destination routers alongwith traffic weights. NetworkX is used to construct and represent the network graph, while Matplotlib handles the visualization. The graph clearly displays router relationships and weighted traffic flows. It aids in understanding high-traffic routes and optimizing telecom infrastructure. The approach supports efficient analysis of large-scale telecom datasets.

**Source code:**

|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  import pandas as pd  df = pd.read\_excel("BSNL\_MPPLAB\_Traffic\_March\_2025.xlsx")  df.head()  source = df["Router1"].dropna().to\_list()  destination = df["Router2"].dropna().to\_list()  pairs = []  for i in range(len(source)):  pairs.append((source[i], destination[i]))  G = nx.Graph()  for router in source:  G.add\_node(router, label=router)  for edge in pairs:  G.add\_edge(edge[0], edge[1])  router\_pos = pd.read\_excel("Router\_Coordinates\_Jittered\_Final.xlsx")  from sklearn.preprocessing import StandardScaler  import numpy as np  scaler = StandardScaler()  router\_positions = {}  routers = router\_pos["From\_Router"].tolist() + router\_pos["To\_Router"].tolist()  router\_latitudes = scaler.fit\_transform(router\_pos[["From\_Latitude"]]).tolist() + scaler.fit\_transform(router\_pos[["To\_Latitude"]]).tolist()  router\_longitudes = scaler.fit\_transform(router\_pos[["From\_Longitude"]]).tolist() + scaler.fit\_transform(router\_pos[["To\_Longitude"]]).tolist()  for i in range(len(routers)):  router\_positions[routers[i]] = (router\_longitudes[i][0], router\_latitudes[i][0])  plt.figure(figsize=(20, 15))  nx.draw(G, pos=router\_positions, with\_labels=True, node\_color='skyblue', node\_size=1000, font\_size=8, font\_weight='bold', edge\_color='gray')  plt.title("Telecom Router Traffic Network (Graph View)")  plt.show() |

**Output:**

The output graph resembles **Fig. 4(a)**, showing the Logical 1 topology of the BSNL Network. Major hubs like Delhi, Mumbai, Hyderabad, Bengaluru-Chennai, and Kolkata are interconnected with color-coded edges. The layout highlights regional traffic flow and router density across India. It provides a clear, scalable view of inter-city telecom connectivity.

**Input Files Format the Network flow modules**

The input format for the Network-Flow module is divided into two types: **Network Topology** and **Demand Matrix**.  
**Input-1 (NetworkInput\_BSNL.dat)** defines the graph with the number of nodes and edges, including the cost and capacity for each edge.  
**Input-2 (FlowInput\_BSNL.dat)** specifies the number of flow demands (commodities) along with their source, destination, and required flow.  
These inputs together help initialize the telecom traffic graph and optimize data flow across the network.  
The given format allows modeling real-time scenarios like telecom traffic management efficiently.

**Input-1 Format(NetworkInput\_BSNL.dat):** <no of vertices><no of edges>

<Source><Destination><Cost><Capacity>

|  |
| --- |
| 705 3164    1 317 0.351000 10000000000  1 501 0.024000 10000000000  1 618 0.311000 10000000000  2 281 0.012000 2500000000  3 281 0.734667 15000000000  3 410 0.907000 10000000000  .  .  .  .  . |

**Source code:**

|  |
| --- |
| import pandas as pd  routers\_file = "sorted\_routers\_list.txt"  connections\_file = "connections\_with\_serial.txt"  csv\_file = "BSNL\_MPPLAB\_Traffic\_March\_2025\_Subset.csv"  output\_dat\_file = "NetworkInput\_BSNL.dat"  router\_to\_id = {}  id\_to\_router = {}  with open(routers\_file, "r") as f:  for line in f:  parts = line.strip().split(maxsplit=1)  if len(parts) == 2:  router\_id, router\_name = parts  router\_id = int(router\_id)  router\_to\_id[router\_name] = router\_id  id\_to\_router[router\_id] = router\_name  num\_routers = len(router\_to\_id)  edges = []  with open(connections\_file, "r") as f:  next(f)  for line in f:  r1\_id, r2\_id = map(int, line.strip().split())  edges.append((r1\_id, r2\_id))  df = pd.read\_csv(csv\_file, usecols=["Router1", "Router2", "Cost", "Capacity"])  df = df.dropna(subset=["Router1", "Router2", "Cost", "Capacity"])  df["Cost"] = pd.to\_numeric(df["Cost"], errors="coerce")  df["Capacity"] = pd.to\_numeric(df["Capacity"], errors="coerce") \* 1\_000\_000\_000  df = df.dropna()  cost\_capacity\_lookup = {}  for \_, row in df.iterrows():  r1, r2 = row["Router1"], row["Router2"]  cost, capacity = row["Cost"], row["Capacity"]  cost\_capacity\_lookup[(r1, r2)] = (cost, capacity)  cost\_capacity\_lookup[(r2, r1)] = (cost, capacity)  final\_edges = []  for r1\_id, r2\_id in edges:  r1\_name = id\_to\_router.get(r1\_id, "")  r2\_name = id\_to\_router.get(r2\_id, "")  cost, capacity = cost\_capacity\_lookup.get((r1\_name, r2\_name), (0.0, 0))  final\_edges.append((r1\_id, r2\_id, cost, capacity))  final\_edges.append((r2\_id, r1\_id, cost, capacity))  final\_edges.sort(key=lambda x: (x[0], x[1]))  num\_edges = len(final\_edges)  with open(output\_dat\_file, "w") as f:  f.write(f"{num\_routers} {num\_edges}\n")  for src, dst, cost, capacity in final\_edges:  f.write(f"{src:>4} {dst:>4} {cost:>10.6f} {capacity:>12.0f}\n") |

**Input-2 Format (FlowInput\_BSNL.dat):**<no of commodities>

<source><destination><flow>

|  |
| --- |
| 10  80 317 0  317 80 0  182 410 0  410 182 0  317 480 0  480 317 0  410 524 0  524 410 0  480 524 0  524 480 0 |

**Vertex Information For Telecom Network Graph**

The files output\_edges.txt and output\_vertices.txt together define the structure of the telecom traffic network.  
output\_edges.txt provides detailed information about the edges, including each city's ID, name, degree (number of direct connections), and a list of adjacent edges, outlining how routers are interconnected via links.  
On the other hand, output\_vertices.txt focuses on the vertex-level perspective, listing each router’s ID, city name, degree, and the adjacent vertices it connects to, representing the network in terms of node-to-node relationships.

**Source code:**

|  |
| --- |
| from collections import defaultdict  # Step 1: Read sorted\_routers\_list.txt to map city IDs to names  city\_names = {}  with open('sorted\_routers\_list.txt', 'r') as f:  for line in f:  parts = line.strip().split(' ', 1)  if len(parts) == 2:  city\_id, name = int(parts[0]), parts[1]  city\_names[city\_id] = name  # Step 2: Read edgenumbers.txt to extract coordinates (not used in output, but kept for completeness)  city\_coords = {}  with open('edgenumbers.txt', 'r') as f:  lines = f.readlines()  for line in lines[1:]: # Skip the first line (header)  parts = line.strip().split()  if len(parts) >= 9:  v1, v2 = int(parts[1]), int(parts[5])  lat1, lon1 = float(parts[3]), float(parts[4])  lat2, lon2 = float(parts[7]), float(parts[8])  if v1 not in city\_coords:  city\_coords[v1] = (lat1, lon1)  if v2 not in city\_coords:  city\_coords[v2] = (lat2, lon2)  # Step 3: Read NetworkInput\_BSNL.txt to build adjacency list and degrees (directed graph)  degrees = defaultdict(int) # Out-degree  adj\_vertices = defaultdict(set) # Outgoing neighbors  adj\_edges = defaultdict(set) # Outgoing edges (edgeno)  with open('NetworkInput\_BSNL.txt', 'r') as f:  lines = f.readlines()  for i, line in enumerate(lines[1:], start=1): # Skip first line (705 3164)  parts = line.strip().split()  if len(parts) >= 2:  v1, v2 = int(parts[0]), int(parts[1])  adj\_vertices[v1].add(v2) # Directed graph: add v2 as an outgoing neighbor of v1  degrees[v1] += 1 # Increment out-degree of v1  adj\_edges[v1].add(i) # Assign edgeno based on line number  # Step 4: Write output\_vertices.txt with S.NO, City\_Id, City\_Name, Degree, Adjacent\_Vertices  with open('output\_vertices.txt', 'w') as f:  header = f"{'S.NO':<6} {'City\_Id':<10} {'City\_Name':<20} {'Degree':<8} {'Adjacent\_Vertices':<60}\n"  f.write(header)  for serial\_no, city\_id in enumerate(range(1, 706), start=1):  name = city\_names.get(city\_id, 'UNKNOWN')  degree = degrees[city\_id]  vertices = sorted(list(adj\_vertices[city\_id]))  vertices\_str = ' '.join(map(str, vertices)) if vertices else ''  line = f"{serial\_no:<6} {city\_id:<10} {name:<20} {degree:<8} {vertices\_str:<60}\n"  f.write(line)  # Step 5: Write output\_edges.txt with S.NO, City\_Id, City\_Name, Degree, Adjacent\_Edges  with open('output\_edges.txt', 'w') as f:  header = f"{'S.NO':<6} {'City\_Id':<10} {'City\_Name':<20} {'Degree':<8} {'Adjacent\_Edges':<60}\n"  f.write(header)  for serial\_no, city\_id in enumerate(range(1, 706), start=1):  name = city\_names.get(city\_id, 'UNKNOWN')  degree = degrees[city\_id]  edges = sorted(list(adj\_edges[city\_id]))  edges\_str = ' '.join(map(str, edges)) if edges else ''  line = f"{serial\_no:<6} {city\_id:<10} {name:<20} {degree:<8} {edges\_str:<60}\n"  f.write(line) |

**Output Format-1 (output\_edges.txt):**

|  |
| --- |
| S.NO City\_Id City\_Name Degree Adjacent\_Edges  1 1 ADILABAD-PE-02 3 1 2 3  2 2 AGARTALA-PE-02 1 4  3 3 AGARTALA-PP-01 2 5 6  4 4 AGRA-PE-02 2 7 8  5 5 AGRA-PE-03 2 9 10  6 6 AGRA-PE-04 2 11 12  7 7 AGRA-PP-02 22 13 14 15 16 17 18 19 20 21 22 23 24  25 26 27 28 29 30 31 32 33 34  .  .  .  . |

**Output Format-2 (output\_vertices.txt):**

|  |
| --- |
| S.NO City\_Id City\_Name Degree Adjacent\_Vertices  1 1 ADILABAD-PE-02 3 317 501 618  2 2 AGARTALA-PE-02 1 281  3 3 AGARTALA-PP-01 2 281 410  4 4 AGRA-PE-02 2 7 525  5 5 AGRA-PE-03 2 7 526  6 6 AGRA-PE-04 2 7 532  7 7 AGRA-PP-02 22 4 5 6 29 32 33 87 250 283 285 339 359 361  372 373 437 438 458 460 461 523 524  .  . |

**1.6.3 Edge Number Information**

Detailed information about physical links between routers in the telecom network is provided, where each row represents a unique network edge. It includes an edge number, source and destination router IDs and names, and their respective geographical coordinates (latitude and longitude). This structure enables accurate mapping of router locations and the logical formation of the network graph. Such data is crucial for geographic visualization and analysis of telecom traffic flow across the network.

**Source Code:**

|  |
| --- |
| import pandas as pd  def parse\_routers\_list(file\_path):  vertex\_to\_router = {}  with open(file\_path, 'r') as f:  for line in f:  vertex, router = line.strip().split(maxsplit=1)  vertex\_to\_router[int(vertex)] = router  return vertex\_to\_router  def parse\_network\_input(file\_path):  edges = []  with open(file\_path, 'r') as f:  next(f)  for line in f:  parts = line.strip().split()  edges.append((int(parts[0]), int(parts[1])))  return edges  def load\_coordinates(excel\_path):  df = pd.read\_excel(excel\_path)  coord\_dict = {}  for \_, row in df.iterrows():  router, lat, lon = row['From\_Router'], row['From\_Latitude'], row['From\_Longitude']  coord\_dict[router] = (lat, lon)  for \_, row in df.iterrows():  router, lat, lon = row['To\_Router'], row['To\_Latitude'], row['To\_Longitude']  if router not in coord\_dict:  coord\_dict[router] = (lat, lon)  return coord\_dict  def generate\_dat\_file(routers\_list\_path, network\_input\_path, excel\_path, output\_path):  vertex\_to\_router = parse\_routers\_list(routers\_list\_path)  edges = parse\_network\_input(network\_input\_path)  coord\_dict = load\_coordinates(excel\_path)  with open(output\_path, 'w') as f:  header = (  f"{'edgeno':<5} "  f"{'vertex1':<5} "  f"{'router\_name1':<18} "  f"{'from\_Latitude':>12} "  f"{'from\_longtitude':>12} "  f"{'vertex2':<5} "  f"{'router\_name2':<18} "  f"{'to\_latitude':>12} "  f"{'to\_longitude':>12}"  )  f.write(header + "\n")  for i, (v1, v2) in enumerate(edges, 1):  router1 = vertex\_to\_router.get(v1, "UNKNOWN")  router2 = vertex\_to\_router.get(v2, "UNKNOWN")  lat1, lon1 = coord\_dict.get(router1, (0.0, 0.0))  lat2, lon2 = coord\_dict.get(router2, (0.0, 0.0))  line = (  f"{i:<5} "  f"{v1:<5} "  f"{router1:<18} "  f"{lat1:>12.6f} "  f"{lon1:>12.6f} "  f"{v2:<5} "  f"{router2:<18} "  f"{lat2:>12.6f} "  f"{lon2:>12.6f}"  )  f.write(line + "\n")  routers\_list\_path = "sorted\_routers\_list.txt"  network\_input\_path = "NetworkInput\_BSNL.txt"  excel\_path = "Router\_Coordinates\_Jittered\_Final.xlsx"  output\_path = "edgenumbers.dat"  generate\_dat\_file(routers\_list\_path, network\_input\_path, excel\_path, output\_path) |

**Output Format:**

|  |
| --- |
| Edgeno vertex1 router\_name1 from\_Latitude from\_longtitude vertex2 router\_name2 to\_latitude to\_longitude  1 1 ADILABAD-PE-02 19.679738 78.538253 317 HYDERABAD-PP-02 17.366860 78.477796  2 1 ADILABAD-PE-02 19.679738 78.538253 501 NAGPUR-PP-03 21.505547 78.980012  3 1 ADILABAD-PE-02 19.679738 78.538253 618 SECUNDERABAD-PP-01 17.432502 78.501539  4 2 AGARTALA-PE-02 23.823426 91.284420 281 GUWAHATI-PP-02 26.186912 91.755559  5 3 AGARTALA-PP-01 23.831279 91.277952 281 GUWAHATI-PP-02 26.186912 91.755559  6 3 AGARTALA-PP-01 23.831279 91.277952 410 KOLKATA-PP-04 22.580707 88.364174  7 4 AGRA-PE-02 27.170310 78.011118 7 AGRA-PP-02 27.176992 78.004783  .  .  .  . |

**1.7 Description of Various Modules**

### ****Jupyter Notebook****

**Jupyter Notebook** is an interactive web-based platform that combines code,visualizations, and documentation in a single interface. It's ideal for research, prototyping, and demonstration.

In our project, Jupyter Notebooks are used for:

* Writing and executing **Python code** in a step-by-step manner.
* Documenting the **logic and results** of each experiment in line with code.
* Creating **interactive visualizations** and plots using Matplotlib or Plotly.
* Demonstrating **data cleaning, transformation, and model results** in a clear and educational format.
* Facilitating **collaboration**, allowing others to understand and reproduce the project findings easily.

**Python**

**Python** is the backbone of our project due to its simplicity, flexibility, and vast library support. Python’s syntax is clean and readable, which makes it suitable for both quick prototypes and complex systems.

Key reasons for choosing Python include:

* Support for **scientific computing** (NumPy, SciPy), **data analysis** (Pandas), **machine learning** (scikit-learn), and **network analysis** (NetworkX).
* Easy integration with **graphical tools** and external APIs.
* Ability to use **multi-threading and multiprocessing** for parallel execution.
* A vast number of **community-contributed packages**, making it easier to build features without starting from scratch

### ****NetworkX****

**NetworkX** is a comprehensive Python library for the creation, manipulation, and study of the structure and dynamics of complex networks.

In our project, NetworkX plays a critical role by:

* Modeling telecom or financial networks as **graph structures** (nodes and edges).
* Performing **graph algorithms** such as shortest path, node centrality, clustering, and flow analysis.
* Visualizing network topologies with built-in functions or integration with external visualization libraries.
* Enabling **weight-based analysis** (e.g., analyzing network capacities, costs, or time delays).
* Supporting **both directed and undirected graphs**, which helps in modeling real-world asymmetric relationships (e.g., one-way communication or transactions)

### ****Pandas****

**Pandas** is an essential Python library used for data manipulation and analysis. It provides powerful and easy-to-use structures such as **DataFrames** and **Series** for handling structured data.

In our project, Pandas is used to:

* **Load and preprocess** large datasets from CSV, Excel, SQL databases, etc.
* Perform **data cleaning**, such as handling missing values, filtering rows, renaming columns, and merging datasets.
* Conduct **statistical summaries**, group operations, and pivot tables.
* Prepare data for graph analysis by converting DataFrames into edge lists or adjacency matrices for NetworkX.
* Generate intermediate insights, trends, and tabular results used in visualizations or reports.

Matplotlib (matplotlib.pyplot)

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Visualize the network graph created using NetworkX.

Define figure size, node colors, edge colors, and text styles for a clear and informative graph.

Customize visual elements such as font size, node size, and graph title to improve readability.

Display the plotted telecom router graph using plt.show() function.

**Scikit-learn (sklearn.preprocessing.StandardScaler)**

Scikit-learn is a widely-used machine learning library in Python. Within it, the StandardScaler module is used for feature scaling and normalization.

Normalize router geographical coordinates (latitude and longitude) to bring them onto the same scale.

Ensure uniformity of coordinate values to prevent plotting distortions during graph visualization.

Create scaled coordinate-based layouts that reflect relative positions of routers more clearly.

**NumPy**

NumPy (Numerical Python) is a core library for numerical computing in Python. It supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.

Support mathematical operations behind data transformations.

Interface with pandas and scikit-learn functions, particularly during coordinate scaling and DataFrame operations.

Provide internal numerical consistency when handling coordinate data for graph plotting.

* 1. **Results and Summary**

The implementation of the **MPPLab Telecom Math Model** using **graph theory** and **Python-based tools** has successfully resulted in the construction and visualization of the **logical topology** of the BSNL network. By modeling routers and their interconnections as nodes and edges in a graph, the project enabled a structured and scalable representation of telecom traffic across India.

#### ****Key Results:****

* The **logical topology** of the BSNL network was derived from a CSV dataset containing router connection information. The topology was represented as an undirected graph using **NetworkX**, with **nodes** symbolizing routers and **edges** representing physical or logical links.
* **Router coordinates** were normalized and used to geographically position the routers for accurate and meaningful network visualization.
* **Matplotlib** enabled the generation of a clear graphical view of the national-level telecom topology, highlighting critical regions such as **Delhi**, **Mumbai**, **Chennai**, **Hyderabad**, and **Kolkata**, which act as major telecom hubs with high traffic density.
* The input files generated for the **Multi-Commodity Flow (MCF)** solver followed standard formats:
  + NetworkInput\_BSNL.dat defined the network graph with edge costs and capacities.
  + FlowInput\_BSNL.dat specified traffic demands (commodities) with source-destination flow requirements.
* Additional utility scripts successfully generated detailed **vertex** and **edge** lists (output\_vertices.txt, output\_edges.txt), listing each city/router’s degree and direct neighbors or links.
* The transformation of **telecom call data to VoIP packet simulation** using the **codec module** and **calls\_to\_voip\_module** further advanced the network modeling to reflect modern IP-based communication systems.

#### ****Overall Summary:****

The project demonstrated a practical integration of **graph theory**, **telecom data analytics**, and **real-time visualization**, providing insights into traffic patterns and network bottlenecks in BSNL’s MPLS infrastructure. By using a multi-core Ubuntu system and tools such as **Python**, **Pandas**, **NetworkX**, and **Jupyter**, the implementation achieved:

* Efficient data processing,
* Scalable network representation,
* Realistic simulation of network load and routing strategies.

**1.9 Limitations**

Despite the successful implementation of the MPPLab Telecom Math Model, several limitations were observed:

* **Incomplete Data**: Missing router coordinates or traffic details can reduce accuracy.
* **Static Model**: The current model does not simulate real-time traffic changes or link failures.
* **Routing Assumptions**: Real BSNL networks often use Single Path Routing, while the model assumes flow splitting (MCF).
* **Fixed Codec Parameters**: VoIP packet conversion uses constant codec settings, which may not reflect actual network variability.
* **Simplified Topology**: City-level abstraction may overlook intra-city router complexities.
* **No Real-Time Integration**: The model isn’t connected to live BSNL data for dynamic validation.
* **Visualization Limits:** Large-scale graphs can be cluttered or slow; advanced tools may be needed for better clarity.