Exp. No.: 1 **QAM MODULATION SISO**

AIM:

To design and simulate QAM Modulation SISO using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Initialize Environment: Clear the command window and any previously stored variables.

2. Set Parameters: Define the number of data points (`N`), the size of the signal constellation

(`mlevel`), and calculate the number of bits per symbol (`k`).

3. Generate Random Bits: Create a random binary bit stream of length `N`.

4. Form Symbols from Bits: Reshape the binary stream into `k`-bit groups and convert these groups

to decimal symbols using binary-to-decimal conversion.

5. Modulate Using QAM: Apply Quadrature Amplitude Modulation (QAM) to the symbols to

generate the modulated signal ready for transmission.

6. Add Noise: Introduce Additive White Gaussian Noise (AWGN) to the transmitted signal using

a specified Signal-to-Noise Ratio (SNR).

7. Demodulate Signal: Demodulate the received noisy signal back into symbols using QAM

demodulation.

8. Convert Symbols to Bits: Convert the demodulated symbols back into a binary bit stream.

9. Calculate Bit Error Rate (BER): Compare the transmitted and received bit streams to calculate

the number of errors and the Bit Error Rate (BER).

10. Visualization: Plot various stages of the modulation process including the transmitted bit

stream, transmitted symbols, constellation diagram of the modulated and received signals, received

symbols, and the received bit stream.

PROGRAM:

% Clear the command window and any variables in the workspace

clear

clc

% Define the number of data points

N = 1000; % number of data

% Define the size of the signal constellation

mlevel = 4; % size of signal constellation

% Calculate the number of bits per symbol

k = log2(mlevel); % number of bits per symbol

% Generate random binary bit stream of length N

x = randi([0 1], N, 1); % signal generation in bit stream

% Convert the bit stream into symbols using binary to decimal conversion

% Reshape the binary stream into a matrix with each column representing a symbol

% and each row representing a set of bits for each symbol

% 'left-msb' indicates that the most significant bit (MSB) is on the left side

xsym = bi2de(reshape(x, k, length(x)/k).', 'left-msb'); % convert the bit stream into symbol stream

% Modulate the symbols using Quadrature Amplitude Modulation (QAM)

xmod = qammod(xsym, mlevel); % modulation

% Store the modulated symbols in a variable for transmission

Tx\_x = xmod;

% Define the Signal-to-Noise Ratio (SNR) in decibels

SNR = 5;

% Add Additive White Gaussian Noise (AWGN) to the transmitted signal

Tx\_awgn = awgn(Tx\_x, SNR, 'measured'); % adding AWGN

% Store the received signal after noise addition

Rx\_x = Tx\_awgn; % Received signal

% Demodulate the received signal to recover the symbols

Rx\_x\_demod = qamdemod(Rx\_x, mlevel); % demodulation

% Convert the demodulated symbols back to binary bits

z = de2bi(Rx\_x\_demod, 'left-msb'); % Convert integers to bits.

% Convert the matrix of bits back to a vector

Rx\_x\_BitStream = reshape(z.', prod(size(z)), 1); % Convert z from a matrix to a vector.

% Calculate the Bit Error Rate (BER) by comparing the transmitted and received bits

[number\_of\_errors, bit\_error\_rate] = biterr(x, Rx\_x\_BitStream); % Calculate BER

% Display BER

disp(['Number of Errors: ', num2str(number\_of\_errors)]);

disp(['Bit Error Rate (BER): ', num2str(bit\_error\_rate)]);

% Plot each step of the process

subplot(5,2,[1 2]); stem(x(1:200),'filled'); title('Transmitted Bit Stream');

subplot(5,2,[3 4]); stem(xsym(1:50),'filled'); title('Transmitted Symbol');

subplot(5,2,5); plot(real(Tx\_x), imag(Tx\_x), 'go', 'MarkerFaceColor', [0, 1, 0]);

axis([-mlevel/2 mlevel/2 -mlevel/2 mlevel/2]);

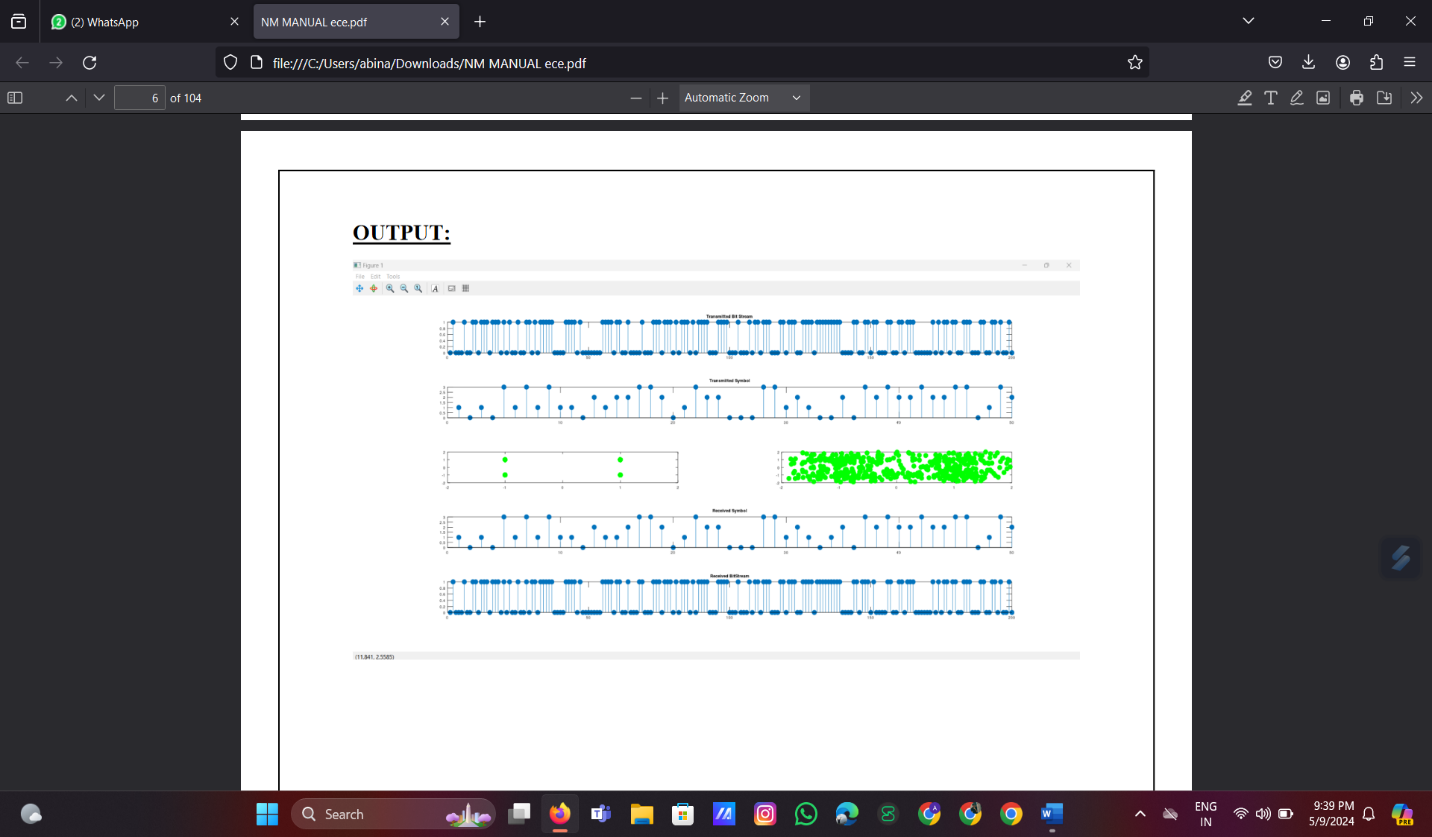
subplot(5,2,6); plot(real(Rx\_x), imag(Rx\_x), 'go', 'MarkerFaceColor', [0, 1, 0]);

axis([-mlevel/2 mlevel/2 -mlevel/2 mlevel/2]);

subplot(5,2,[7 8]); stem(Rx\_x\_demod(1:50),'filled'); title('Received Symbol');

subplot(5,2,[9 10]); stem(Rx\_x\_BitStream(1:200),'filled'); title('Received BitStream');

Output:



**QAM MIMO BER**

PROGRAM:

% Script for computing the Bit Error Rate (BER) for Binary Phase Shift Keying (BPSK)

modulation in a

% Rayleigh fading channel with 2 Transmitters (Tx) and 2 Receivers (Rx) MIMO channel,

% using Minimum Mean Square Error (MMSE) equalization.

clear; % Clear workspace variables

N = 10^6; % Number of bits or symbols

Eb\_N0\_dB = [0:25]; % Multiple Eb/N0 values in dB

nTx = 2; % Number of transmitters

nRx = 2; % Number of receivers

for ii = 1:length(Eb\_N0\_dB)

% Transmitter

% Generate random bit sequence for transmission

ip = rand(1, N) > 0.5; % Generating 0s and 1s with equal probability

s = 2 \* ip - 1; % BPSK modulation: 0 -> -1; 1 -> 0

% Modulate the symbols for MIMO transmission

sMod = kron(s, ones(nRx, 1));

sMod = reshape(sMod, [nRx, nTx, N / nTx]);

% Generate Rayleigh fading channel

h = 1/sqrt(2) \* [randn(nRx, nTx, N / nTx) + j \* randn(nRx, nTx, N / nTx)];

% Generate white Gaussian noise with 0dB variance

n = 1/sqrt(2) \* [randn(nRx, N / nTx) + j \* randn(nRx, N / nTx)];

% Channel and noise addition

y = squeeze(sum(h .\* sMod, 2)) + 10^(-Eb\_N0\_dB(ii) / 20) \* n;

% Receiver

12. BER Calculation:

- Calculate the BER for each \(E\_b/N\_0\) value as the ratio of the number of errors to the total

number of transmitted bits.

13. Theoretical BER Calculation:

- Compute theoretical BER for comparison:

- For a single receiver (nRx=1).

- For two receivers using Maximal Ratio Combining (MRC) in a diversity scenario.

PROGRAM:

% Script for computing the Bit Error Rate (BER) for Binary Phase Shift Keying (BPSK)

modulation in a

% Rayleigh fading channel with 2 Transmitters (Tx) and 2 Receivers (Rx) MIMO channel,

% using Minimum Mean Square Error (MMSE) equalization.

clear; % Clear workspace variables

N = 10^6; % Number of bits or symbols

Eb\_N0\_dB = [0:25]; % Multiple Eb/N0 values in dB

nTx = 2; % Number of transmitters

nRx = 2; % Number of receivers

for ii = 1:length(Eb\_N0\_dB)

% Transmitter

% Generate random bit sequence for transmission

ip = rand(1, N) > 0.5; % Generating 0s and 1s with equal probability

s = 2 \* ip - 1; % BPSK modulation: 0 -> -1; 1 -> 0

% Modulate the symbols for MIMO transmission

sMod = kron(s, ones(nRx, 1));

sMod = reshape(sMod, [nRx, nTx, N / nTx]);

% Generate Rayleigh fading channel

h = 1/sqrt(2) \* [randn(nRx, nTx, N / nTx) + j \* randn(nRx, nTx, N / nTx)];

% Generate white Gaussian noise with 0dB variance

n = 1/sqrt(2) \* [randn(nRx, N / nTx) + j \* randn(nRx, N / nTx)];

% Channel and noise addition

y = squeeze(sum(h .\* sMod, 2)) + 10^(-Eb\_N0\_dB(ii) / 20) \* n;

% Receiver

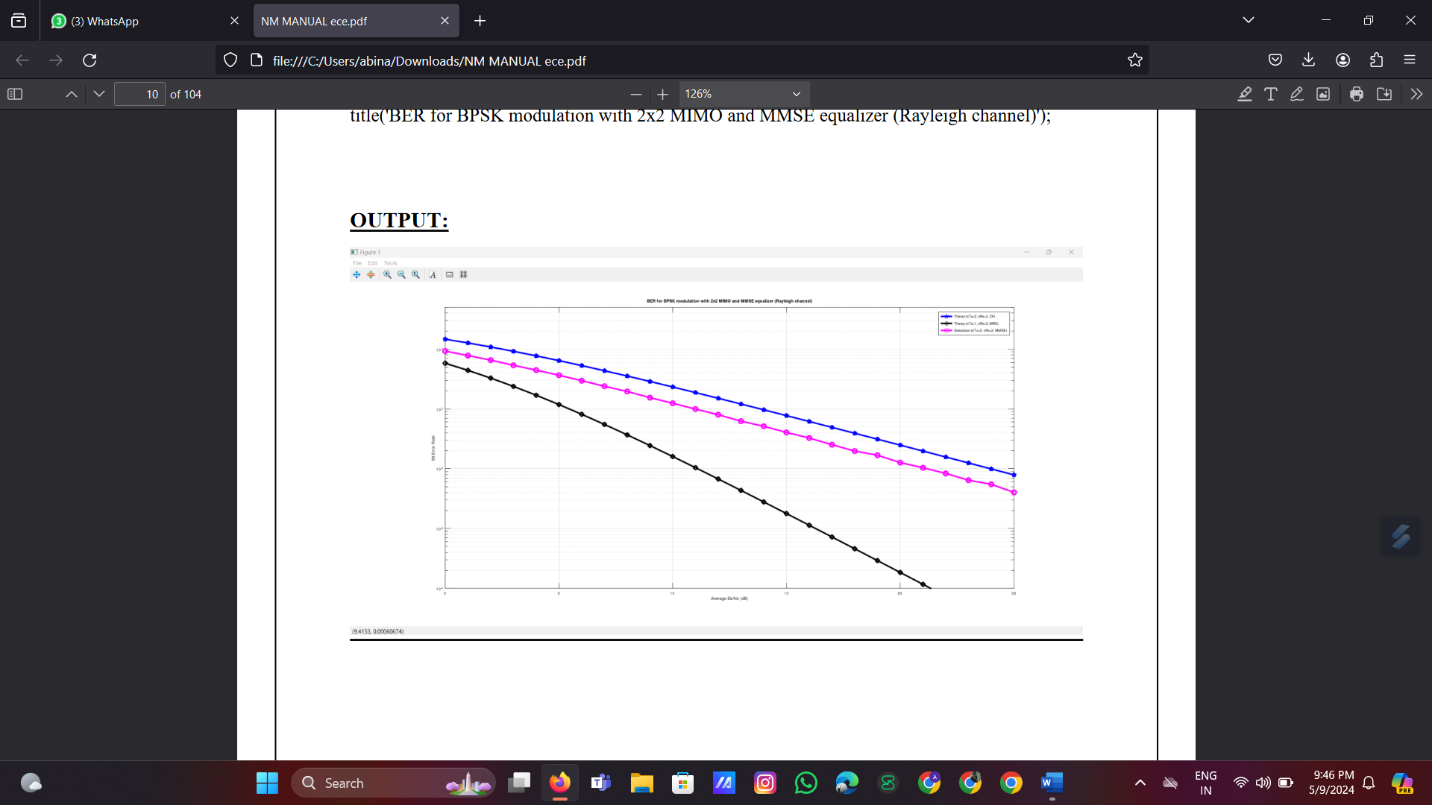
nRx=2, MMSE)');

xlabel('Average Eb/No (dB)');

ylabel('Bit Error Rate');

title('BER for BPSK modulation with 2x2 MIMO and MMSE equalizer (Rayleigh channel)');

output:



Exp. No.: 3 **OFDM WAVE FORM GENERATION**

PROGRAM:

% Clear the command window and any variables in the workspace

clc

clear

% Load the communications package

pkg load communications

% Define the size of the FFT (Fast Fourier Transform)

nFFTSize = 128;

% Define the subcarrier index range for each symbol

% The indices range from -26 to -1 and 1 to 26

subcarrierIndex = [-26:-1 1:26];

% Define the number of bits for transmission

nBit = 2500;

% Generate a random binary sequence of length nBit

ip = rand(1, nBit) > 0.5; % generating 1's and 0's

% Define the number of bits per symbol

nBitPerSymbol = 1;

% Calculate the number of symbols required to transmit nBit

nSymbol = ceil(nBit / nBitPerSymbol);

% Modulate the binary sequence using Binary Phase Shift Keying (BPSK)

% Convert 0's to -1 and 1's to +1

ipMod = 2 \* ip - 1;

% Pad the modulated symbols with zeros to match the required length

ipMod = [ipMod zeros(1, nBitPerSymbol \* nSymbol - nBit)];

% Reshape the modulated symbols into a matrix with each row representing a symbol

% and each column representing bits for each symbol

ipMod = reshape(ipMod, nSymbol, nBitPerSymbol);

% Initialize an empty vector to store the OFDM symbols

st = [];

% Iterate over each symbol

for ii = 1:nSymbol

% Initialize an array to store the input for the Inverse Fast Fourier Transform (IFFT)

inputiFFT = zeros(1, nFFTSize);

% Assign the bits from the modulated symbols to subcarriers

inputiFFT(subcarrierIndex + nFFTSize / 2 + 1) = ipMod(ii, :);

% Shift the subcarriers at indices [-26 to -1] to fft input indices [38 to 63]

inputiFFT = fftshift(inputiFFT);

% Perform IFFT to convert frequency domain symbols to time domain

outputiFFT = ifft(inputiFFT, nFFTSize);

% Add cyclic prefix of 16 samples to the output

outputiFFT\_with\_CP = [outputiFFT(49:64) outputiFFT];

% Concatenate the OFDM symbol with cyclic prefix to the vector st

st = [st outputiFFT\_with\_CP];

end

% Close all open figures

close all

% Define the subcarrier spacing and bandwidth

% 64 subcarrier , with 15KHz subcarrier spacing equals 960KHZ Bandwidth

fsMHz = 1.92; % 960 KHz

nOverlap = 20; % Overlap for pwelch function

% Compute the power spectral density using the Welch method

[Pxx, W] = pwelch(st, [], [], 4096, nOverlap);

% Plot the power spectral density

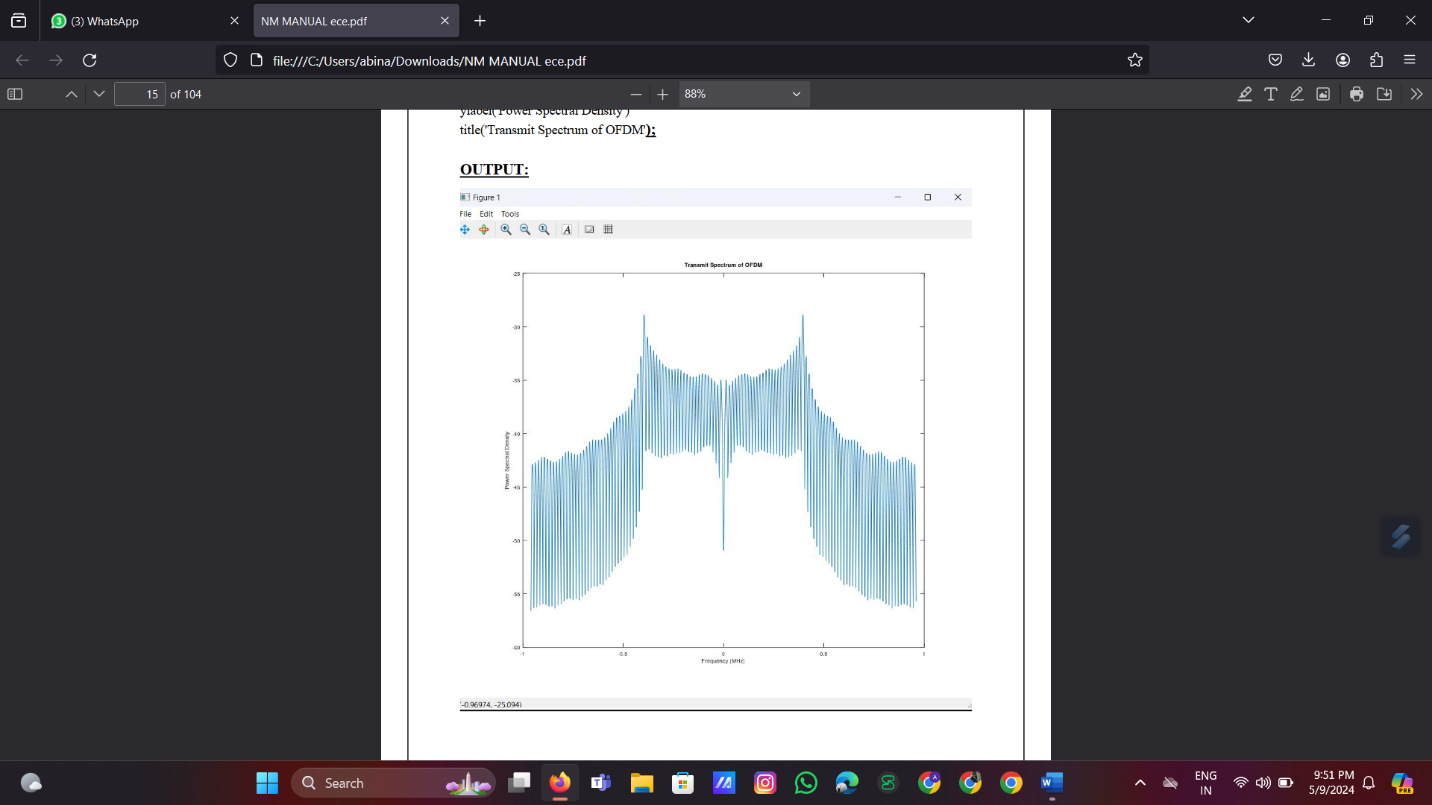
plot([-2048:2047] \* fsMHz / 4096, 10 \* log10(fftshift(Pxx)));

xlabel('Frequency (MHz)')

ylabel('Power Spectral Density')

title('Transmit Spectrum of OFDM');

output:



Exp. No.: 5 **eMBB OPTIMIZATION WORKSHOP**

PROGRAM:  
% Parameters  
num\_users = 50; % Number of mobile users  
cell\_radius = 500; % Radius of the cell (assuming circular cell coverage)  
user\_speed = 30; % Average user speed in km/h  
simulation\_duration = 60; % Simulation duration in seconds  
time\_step = 1; % Time step for simulation in seconds  
% Antenna configuration parameters  
antenna\_height = 25; % Height of the antenna in meters  
antenna\_tilt = 5; % Antenna tilt angle in degrees  
beamwidth = 120; % Antenna beamwidth in degrees  
% Bandwidth optimization parameters  
initial\_bandwidth = 100e6; % Initial bandwidth in Hz  
% Additional parameters for dynamic bandwidth allocation algorithm can be added here  
% Randomly place users within the cell  
user\_positions = cell\_radius \* exp(2i \* pi \* rand(1, num\_users));  
% Simulate user mobility over time  
for t = 0:time\_step:simulation\_duration  
% Update user positions based on their speed and direction  
user\_positions = user\_positions + user\_speed \* (time\_step / 3600) \* exp(2i \* pi \* rand(1,  
num\_users));  
% Plot user positions  
plot(real(user\_positions), imag(user\_positions), 'o');  
title('5G eMBB Optimization - User Mobility');  
xlabel('X-axis');  
ylabel('Y-axis');  
xlim([-cell\_radius, cell\_radius]);  
ylim([-cell\_radius, cell\_radius]);  
drawnow;

Exp. No.: 6

URLLC IMPLEMENTATION CHALLENGE

PROGRAM:

% Define parameters

numDevices = 10; % Number of IoT devices

latencyThreshold = 5; % Latency threshold in milliseconds for URLLC

reliabilityThreshold = 0.95; % Required reliability

numSlices = 3; % Number of network slices

sliceNames = {'URLLC', 'eMBB', 'mMTC'}; % Slice names

QoSValues = [10, 5, 3]; % QoS values for each slice

% Simulate latency for each device

deviceLatencies = randi([1, 10], 1, numDevices); % Random latency between 1 and 10 ms

% Check if latency meets URLLC requirements

urlLCDevices = deviceLatencies <= latencyThreshold;

% Simulate reliability for each device

deviceReliability = rand(1, numDevices); % Random reliability between 0 and 1

% Check if reliability meets requirements

reliableDevices = deviceReliability >= reliabilityThreshold;

% Assign devices to network slices based on QoS requirements

sliceAssignments = zeros(1, numDevices);

for i = 1:numDevices

if urlLCDevices(i)

sliceAssignments(i) = 1; % Assign to URLLC slice

else

[~, sliceIndex] = max(QoSValues); % Find slice with highest QoS value

sliceAssignments(i) = sliceIndex;

end

end

% Display slice assignments

disp('Device Slice Assignments:');

for i = 1:numDevices

disp(['Device ', num2str(i), ' assigned to slice ', sliceNames{sliceAssignments(i)}]);

end

% Analyze impact of QoS settings on latency and reliability

averageLatency = mean(deviceLatencies);

averageReliability = mean(deviceReliability);

disp(['Average Latency: ', num2str(averageLatency), ' ms']);

disp(['Average Reliability: ', num2str(averageReliability)]);

% Plotting

figure;

subplot(2,1,1);

bar(1:numDevices, deviceLatencies);

title('Device Latencies');

xlabel('Device Index');

ylabel('Latency (ms)');

subplot(2,1,2);

bar(1:numDevices, deviceReliability);

title('Device Reliability');

xlabel('Device Index');

ylabel('Reliability');

sgtitle('Latency and Reliability Analysis');

Exp. No.: 8 **5G NETWORK SLICING WORKSHOP**

PROGRAM:

% Parameters

num\_applications = 3; % Number of diverse applications

num\_users\_per\_slice = 20; % Number of users per network slice

% Generate random network slice characteristics for each application

slice\_bandwidths = randi([50, 200], 1, num\_applications); % Bandwidth in Mbps

slice\_latency\_targets = randi([5, 20], 1, num\_applications); % Latency targets in milliseconds

% Simulate network slices for each application

for app = 1:num\_applications

% Generate random user positions within a predefined area for each slice

user\_positions = 100 \* rand(num\_users\_per\_slice, 2);

% Display slice information

disp(['Application ', num2str(app)]);

disp(['Slice Bandwidth: ', num2str(slice\_bandwidths(app)), ' Mbps']);

disp(['Slice Latency Target: ', num2str(slice\_latency\_targets(app)), ' ms']);

% Visualize user positions within the slice

scatter(user\_positions(:, 1), user\_positions(:, 2), 'filled');

title(['5G Network Slicing - Application ', num2str(app)]);

xlabel('X-axis');

ylabel('Y-axis');

xlim([0, 100]);

ylim([0, 100]);

drawnow;

end

Exp. No.: 10 **NETWORK SECURITY**

AIM:

To design and simulate NETWORK SECURITY using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

Step 1 Function Definitions:

- Define functions to simulate secure communication in a 5G network.

- Functions include `secureCommunication` to simulate the communication process,

`encryptMessage` for encryption, `decryptMessage` for decryption, and `generateEncryptionKey`

for key generation.

2. Simulation Setup

- Define the message to be transmitted.

- Encrypt the message using the `encryptMessage` function, simulating confidentiality.

- Decrypt the encrypted message using the `decryptMessage` function to ensure integrity.

Step 3: Display Results:

- Display the original message, encrypted message, and decrypted message to verify the

security mechanisms.

PROGRAM:

% Simulate 5G network communication

function secureCommunication()

% Parameters

message = 'Hello, 5G World!'; % Message to be transmitted

% Security mechanisms

encrypted\_message = encryptMessage(message);

decrypted\_message = decryptMessage(encrypted\_message);

% Display results

disp(['Original Message: ', message]);

disp(['Encrypted Message: ', encrypted\_message]);

disp(['Decrypted Message: ', decrypted\_message]);

end

% Encryption function (simulating confidentiality)

function encrypted\_message = encryptMessage(message)

key = generateEncryptionKey(length(message));

encrypted\_message = char(bitxor(uint8(message), key)); % XOR encryption for ASCII values

end

% Decryption function (simulating confidentiality)

function decrypted\_message = decryptMessage(encrypted\_message)

key = generateEncryptionKey(length(encrypted\_message));

decrypted\_message = char(bitxor(uint8(encrypted\_message), key)); % XOR decryption

end

% Key generation function (simulating secure key generation)

function key = generateEncryptionKey(length\_message)

key\_length = length\_message; % Key length in bits

key = randi([0, 255], 1, length\_message); % Simulating a randomly generated key

end

% Run the simulation

secureCommunication();

**5G BEAM FORMING COVERAGE ENHANCEMENT**

PROGRAM:

% Simulated 5G Network Environment Setup

num\_base\_stations = 5;

coverage\_map = zeros(100, 100); % Assuming a 100x100 grid for coverage map

% Simulate base station locations

base\_station\_locations = randi([1, 100], num\_base\_stations, 2);

% Simulate obstacles that affect signal propagation

num\_obstacles = 10;

obstacle\_locations = randi([1, 100], num\_obstacles, 2);

% Generate random coverage map

for i = 1:num\_base\_stations

coverage\_map(base\_station\_locations(i, 1), base\_station\_locations(i, 2)) = 1;

end

% Add obstacles to coverage map

for i = 1:num\_obstacles

coverage\_map(obstacle\_locations(i, 1), obstacle\_locations(i, 2)) = -1;

end

% Visualize coverage map

figure;

imagesc(coverage\_map);

colorbar;

title('Initial Coverage Map');

xlabel('X-coordinate');

ylabel('Y-coordinate');

% Beamforming Configuration

% Implement simple beamforming by focusing signals towards areas with weak coverage

% Find areas with weak coverage

weak\_coverage\_indices = find(coverage\_map == 0);

% Implement beamforming by adjusting signal strength towards weak coverage areas

for idx = 1:numel(weak\_coverage\_indices)

[x, y] = ind2sub(size(coverage\_map), weak\_coverage\_indices(idx));

% Implement beamforming by increasing signal strength in weak coverage areas

coverage\_map(x, y) = 2; % For simplicity, assuming doubling the signal strength

end

% Visualize coverage map after beamforming

figure;

imagesc(coverage\_map);

colorbar;

title('Coverage Map after Beamforming');

xlabel('X-coordinate');

ylabel('Y-coordinate’);

**5G CORE NETWORK DESIGN CHALLENGE**

PROGRAM:

% Define parameters

latency\_requirements = [10, 20, 30]; % Latency requirements in milliseconds for different types

of traffic

bandwidth\_requirements = [100, 200, 300]; % Bandwidth requirements in Mbps for different

applications

reliability\_threshold = 0.99; % Desired reliability threshold

network\_scale\_factor = 1.5; % Factor for network scalability

% Define core network elements

num\_base\_stations = 10;

num\_users\_per\_station = 5;

num\_UPF = 2;

num\_SMF = 1;

num\_AMF = 1;

% Simulation: Generate random throughput and interference matrices

throughput = rand(num\_base\_stations, num\_users\_per\_station) \* 100; % Random throughput

values (Mbps)

interference\_matrix = rand(num\_base\_stations, num\_base\_stations);

% Optimization: Implement optimization strategies (placeholder)

% Here, you can implement optimization algorithms for resource allocation, routing, and traffic

management

% Quality of Service (QoS) Implementation (placeholder)

% Configure QoS parameters to prioritize traffic based on requirements

qos\_params = struct('priority', [1, 2, 3], 'traffic\_type', {'VoIP', 'Video Streaming', 'Web Browsing'},

'latency\_requirement', latency\_requirements, 'bandwidth\_requirement', bandwidth\_requirements);

% Security Integration (placeholder)

% Integrate security measures such as encryption, access control, and authentication protocols

security\_protocols = {'Encryption', 'Access Control', 'Authentication'};

% Calculate total throughput of the network

total\_throughput = sum(sum(throughput));

% Plot throughput distribution across base stations

figure;

bar(1:num\_base\_stations, sum(throughput, 2), 'b');

xlabel('Base Station');

ylabel('Total Throughput (Mbps)');

title('Throughput Distribution Across Base Stations');

grid on;

% Display results

disp(['Total throughput of the network: ' num2str(total\_throughput) ' Mbps']);

disp(' ');

% Display QoS parameters

disp('Quality of Service (QoS) Parameters:');

disp(qos\_params);

disp(' ');

% Display Security Protocols

disp('Security Protocols:');

disp(security\_protocols);

**5G TELEMEDICINE DEPLOYMENT CHALLENGE**

PROGRAM:

% Define network infrastructure

network\_location = [0, 0];

medical\_device\_locations = [

1, 1; % Example medical device 1

-1, 1; % Example medical device 2

0, -1 % Example medical device 3

];

optimized\_delays = [5, 4, 3]; % Optimized delays for each device in milliseconds

video\_data\_rates = [10, 8, 12]; % Data transfer rates for video consultation in Mbps

% Plot network infrastructure

scatter(network\_location(1), network\_location(2), 200, 'b', 'filled');

text(network\_location(1), network\_location(2), '5G Network', 'HorizontalAlignment', 'center');

hold on;

% Plot medical devices

scatter(medical\_device\_locations(:, 1), medical\_device\_locations(:, 2), 100, 'r', 'filled');

for i = 1:size(medical\_device\_locations, 1)

text(medical\_device\_locations(i, 1), medical\_device\_locations(i, 2), sprintf('Device %d', i),

'HorizontalAlignment', 'center');

end

% Draw optimized transmission paths between network and devices

for i = 1:size(medical\_device\_locations, 1)

line([network\_location(1), medical\_device\_locations(i, 1)], [network\_location(2),

medical\_device\_locations(i, 2)], 'Color', 'k', 'LineStyle', '-');

text(medical\_device\_locations(i, 1), medical\_device\_locations(i, 2) + 0.2, sprintf('Delay: %d

ms', optimized\_delays(i)), 'HorizontalAlignment', 'center', 'Color', 'b');

text((network\_location(1) + medical\_device\_locations(i, 1)) / 2, (network\_location(2) +

medical\_device\_locations(i, 2)) / 2 - 0.2, sprintf('Data Rate: %d Mbps', video\_data\_rates(i)),

'HorizontalAlignment', 'center', 'Color', 'r');

end

% Print security measures and compliance checks data

disp('Security Measures:');

disp('- Encrypt data transmission');

disp('- Authenticate devices and users');

disp('Compliance Checks:');

disp('- Ensure compliance with healthcare data protection regulations');

xlabel('X Position');

ylabel('Y Position');

title('Simulated Telemedicine Environment Deployment with Minimized Delays and Video DataRates');

axis equal;

grid on;

5G – ENABLED ENVIRONMENTAL MONITORING

IMPLEMENTATION CHALLENGE

PROGRAM:

function data = useCase17()

% Simulating sensor data collection for temperature, humidity, and air quality

temperature = randi([20, 30], 1, 1); % Random temperature between 20 to 30 degrees Celsius

humidity = randi([40, 60], 1, 1); % Random humidity between 40% to 60%

air\_quality = randi([1, 100], 1, 1); % Random air quality index between 1 to 100

% Combine sensor readings into a matrix

data = [temperature, humidity, air\_quality];

% Real-Time Data Transmission

% Simulating data transmission over 5G network

fprintf('Transmitting data over 5G network: %s\n', mat2str(data));

% Actual implementation would involve sending data to a server or cloud platform

% Data Processing and Analytics

% Simulating basic data processing by calculating average values

average\_temperature = mean(data(:, 1));

average\_humidity = mean(data(:, 2));

average\_air\_quality = mean(data(:, 3));

% Combine processed data into a matrix

processed\_data = [average\_temperature, average\_humidity, average\_air\_quality];

% Alerting Mechanism

% Simulating sending alert message

if processed\_data(3) > 70 % If air quality is poor

fprintf('Sending Alert: Alert: Poor air quality detected!\n');

% Actual implementation would involve sending alerts via email, SMS, or push notifications

end

% Visualization Interface

% Plotting real-time environmental data

figure;

subplot(3, 1, 1);

plot(data(:, 1), 'r', 'LineWidth', 2);

xlabel('Time');

ylabel('Temperature (°C)');

title('Real-Time Environmental Data');

subplot(3, 1, 2);

plot(data(:, 2), 'g', 'LineWidth', 2);

xlabel('Time');

ylabel('Humidity (%)');

subplot(3, 1, 3);

plot(data(:, 3), 'b', 'LineWidth', 2);

xlabel('Time');

ylabel('Air Quality Index');

legend('Temperature', 'Humidity', 'Air Quality');

end

% Call the function to start the process

**NETWORK PERFORMANCE MONITORING AND OPTIMIZATION CHALLENGE**

PROGRAM:

function simulate\_5G\_network()

% Generate random performance metrics

metrics = rand(1, 10) \* 100; % Example: 10 random metrics

% Set thresholds

threshold = 70; % Example: Threshold set to 70

% Display metrics

disp("Performance Metrics:");

disp(metrics);

% Plot metrics and threshold

figure;

plot(metrics, 'b.-');

hold on;

plot([1, length(metrics)], [threshold, threshold], 'r--');

xlabel('Metric Index');

ylabel('Metric Value');

title('5G Network Performance Metrics');

legend('Metrics', 'Threshold');

hold off;

% Check metrics against threshold

for i = 1:length(metrics)

if metrics(i) > threshold

disp(['Alert: Metric ', num2str(i), ' crossed threshold at ', num2str(metrics(i))]);

% Call troubleshooting function

troubleshooting(i, metrics(i));

end

end

% Propose optimization strategy

disp("Optimization Strategy: Implement load balancing to distribute traffic evenly.");

end

function troubleshooting(metric\_index, metric\_value)

% Simulate troubleshooting scenarios based on the metric crossing threshold

switch metric\_index

case 1

disp("Troubleshooting Scenario: High latency detected.");

disp("Action: Check for interference, adjust antenna orientation.");

case 2

disp("Troubleshooting Scenario: High packet loss detected.");

disp("Action: Check for network congestion, optimize routing.");

otherwise

disp("Troubleshooting Scenario: Unknown issue detected.");

disp("Action: Perform general network health check.");

end

end

% Main function call

simulate\_5G\_network();