## PROGRAM CODE-4

**MQTT TOPIC CONFIGURATION USING SINGLE TOPIC**

## Section 1: #!/usr/bin/env python3

|  |
| --- |
| #!/usr/bin/env python3  import rclpy from rclpy.node import Node from std\_msgs.msg import Float32, String import socket import struct |

## Section 2: class UDPServerNode(Node):

|  |
| --- |
| class UDPServerNode(Node):  def \_\_init\_\_(self):  super().\_\_init\_\_('udp\_server') |

## Section 3: # Configuration

|  |
| --- |
| # Configuration  self.udp\_ip = "172.16.11.12"  self.udp\_port = 7000  self.destination\_ip = "172.16.11.70"  self.destination\_port = 7000 |

## Section 4: # Initialize UDP socket

|  |
| --- |
| # Initialize UDP socket  self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  self.sock.bind((self.udp\_ip, self.udp\_port))   self.get\_logger().info(f"UDP Server listening on {self.udp\_ip}:{self.udp\_port}") |

## Section 5: # Send initial message

|  |
| --- |
| # Send initial message  self.sock.sendto("Connection Established".encode(), (self.destination\_ip, self.destination\_port)) |

## Section 6: # ROS 2 Publisher (ROS topic 1: publish UDP data to MQTT via bridge)

|  |
| --- |
| # ROS 2 Publisher (ROS topic 1: publish UDP data to MQTT via bridge)  self.data\_publisher = self.create\_publisher(String, '/ros2mqtt/frequency', 10) |

## Section 7: # ROS 2 Subscriber (ROS topic 2: subscribe to data from MQTT via bridge)

|  |
| --- |
| # ROS 2 Subscriber (ROS topic 2: subscribe to data from MQTT via bridge)  self.data\_subscriber = self.create\_subscription(  String,  '/mqtt2ros/frequency',  self.subscriber\_callback,  10  ) |

## Section 8: # Start receiving UDP messages

|  |
| --- |
| # Start receiving UDP messages  self.timer = self.create\_timer(0.05, self.receive\_message) # Check every 0.05s (20 Hz)   def receive\_message(self):  try:  data, addr = self.sock.recvfrom(4) # Expect 4 bytes for a float  self.get\_logger().info(f"Received raw data from {addr}: {data}") |

## Section 9: # Try to interpret data as a float

|  |
| --- |
| # Try to interpret data as a float  try:  received\_float = struct.unpack('f', data)[0]  string\_msg = String()  string\_msg.data = f"UDP received: {received\_float}"  self.data\_publisher.publish(string\_msg)  self.get\_logger().info(f"msg Published to /ros2mqtt/frequency: {received\_float}")  except struct.error:  self.get\_logger().warn(f"Invalid float data from {addr}: {data}")   except Exception as e:  self.get\_logger().error(f"Error receiving data: {e}")   def subscriber\_callback(self, msg):  """Callback for the second ROS topic subscription."""  self.get\_logger().info(f"Received from /mqtt2ros/frequency: {msg.data}")   def destroy\_node(self):  self.get\_logger().info("Shutting down UDP server...")  self.sock.close()  super().destroy\_node() |

## Section 10: def main(args=None):

|  |
| --- |
| def main(args=None):  rclpy.init(args=args)  node = UDPServerNode()  try:  rclpy.spin(node)  except KeyboardInterrupt:  node.get\_logger().info("Keyboard Interrupt detected, shutting down.")  finally:  node.destroy\_node()  rclpy.shutdown()  if \_\_name\_\_ == '\_\_main\_\_':  main() |