## PROGRAM CODE-5

**UDP node Initialization using single topics**

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

|  |
| --- |
| #!/usr/bin/env python3  import rclpy from rclpy.node import Node import socket import struct import paho.mqtt.client as mqtt |

## Section 2: class UDPServerNode(Node):

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

## Section 3: # UDP Configuration

|  |
| --- |
| # UDP 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: # MQTT Configuration

|  |
| --- |
| # MQTT Configuration  self.mqtt\_broker = "172.16.11.70"  self.mqtt\_port = 1883  self.mqtt\_topic = "frequency" # Updated MQTT topic |

## Section 7: # Initialize MQTT client

|  |
| --- |
| # Initialize MQTT client  self.mqtt\_client = mqtt.Client(client\_id="udp\_server\_node")  self.mqtt\_client.on\_connect = self.on\_mqtt\_connect  self.mqtt\_client.on\_message = self.on\_mqtt\_message  self.mqtt\_client.connect(self.mqtt\_broker, self.mqtt\_port)  self.mqtt\_client.loop\_start() |

## 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 on\_mqtt\_connect(self, client, userdata, flags, rc):  """Callback for when the MQTT client connects to the broker."""  if rc == 0:  self.get\_logger().info(f"Connected to MQTT broker at {self.mqtt\_broker}:{self.mqtt\_port}")  self.mqtt\_client.subscribe(self.mqtt\_topic)  self.get\_logger().info(f"Subscribed to MQTT topic: {self.mqtt\_topic}")  else:  self.get\_logger().error(f"Failed to connect to MQTT broker, return code: {rc}")   def on\_mqtt\_message(self, client, userdata, msg):  """Callback for when a message is received from the MQTT topic."""  self.get\_logger().info(f"Received from MQTT topic {msg.topic}: {msg.payload.decode()}")   def receive\_message(self):  """Receive UDP data and publish to MQTT topic."""  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]  mqtt\_message = f"UDP received: {received\_float}"  self.mqtt\_client.publish(self.mqtt\_topic, mqtt\_message)  self.get\_logger().info(f"Published to MQTT topic {self.mqtt\_topic}: {mqtt\_message}")  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 UDP data: {e}")   def destroy\_node(self):  """Clean up resources on node shutdown."""  self.get\_logger().info("Shutting down UDP server...")  self.sock.close()  self.mqtt\_client.loop\_stop()  self.mqtt\_client.disconnect()  self.get\_logger().info("Disconnected from MQTT broker")  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() |