## LAB EXERCISES

### Experiment 1: Setting up a Simple Wireless Sensor Network

- **Objective**: Set up a simple wireless sensor network using a simulator (e.g., NS2, NS3, or OMNeT++).
- **Tools Required**:
  - Simulator: NS2, NS3, or OMNeT++ (NS3 is used in this example)
  - Hardware: None
- **Procedure**:

  0. **Install NS3 Simulator**:
     - Download NS3 from the official website.
     - Follow the installation guide specific to your operating system.
  1. **Create the Simulation Script**:
     - Write a simulation script in C++ to define the wireless sensor network, its nodes, mobility, and communication.
  2. **Define Nodes**:
     - Create a set of sensor nodes in the network.
     - Use a grid or random topology for node placement.
  3. **Setup Wireless Communication**:
     - Use IEEE 802.11 or ZigBee communication standards.
     - Define range, data rates, and energy models.
  4. **Run the Simulation**:
     - Execute the script using NS3's simulation engine.
     - Analyze the generated logs, traces, or visual outputs.

- **Sample Code (NS3 Simulation Script)**:

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/wifi-module.h"
#include "ns3/energy-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

int main() {
   // Set up logging for debugging
   LogComponentEnable("WifiSimpleAdhocGrid", LOG_LEVEL_INFO);

   // Create nodes for the network
   NodeContainer sensorNodes;
   sensorNodes.Create(10);

   // Set up mobility model
   MobilityHelper mobility;
   mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                          "MinX", DoubleValue(0.0),
                          "MinY", DoubleValue(0.0),
                          "DeltaX", DoubleValue(5.0),
                          "DeltaY", DoubleValue(5.0),
                                "GridWidth", UintegerValue(3),
                                "LayoutType", StringValue("RowFirst"));

    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(sensorNodes);

    // Install Wi-Fi devices
    WifiHelper wifi;
    wifi.SetStandard(WIFI_PHY_STANDARD_80211b);

    YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
    YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
    wifiPhy.SetChannel(wifiChannel.Create());

    WifiMacHelper wifiMac;
    wifiMac.SetType("ns3::AdhocWifiMac");

    NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, sensorNodes);

    // Install Internet stack
    InternetStackHelper internet;
    internet.Install(sensorNodes);

    // Assign IP addresses
    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = ipv4.Assign(devices);

    // Set up a UDP echo server on node 0
    uint16_t port = 9;
    UdpEchoServerHelper echoServer(port);
    ApplicationContainer serverApp = echoServer.Install(sensorNodes.Get(0));
    serverApp.Start(Seconds(1.0));
    serverApp.Stop(Seconds(10.0));

    // Set up a UDP echo client on node 1
    UdpEchoClientHelper echoClient(interfaces.GetAddress(0), port);
    echoClient.SetAttribute("MaxPackets", UintegerValue(2));
    echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    echoClient.SetAttribute("PacketSize", UintegerValue(1024));

    ApplicationContainer clientApp = echoClient.Install(sensorNodes.Get(1));
    clientApp.Start(Seconds(2.0));
    clientApp.Stop(Seconds(10.0));

    // Run the simulation
    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

- **Key Points in the Script**:
    1. **Node Creation**: A total of 10 nodes are created.
    2. **Mobility**: Nodes are arranged in a grid layout with specified spacing.

3. **Wireless Configuration**: Wi-Fi devices are configured for ad hoc communication.
4. **Application**: A UDP echo application is set up to demonstrate data communication.

- **Execution Steps**:
  1. Save the code as simple-wsn.cc.
  2. Compile the code using the waf build system:
  3. ./waf configure
  4. ./waf build
  5. ./waf --run simple-wsn
  6. Analyze the output logs or use a visualization tool (e.g., NetAnim) to observe node communication.
- **Expected Outcome**:
  - Successful setup of a wireless sensor network with 10 nodes.
  - Communication between nodes, verified through the UDP echo application.
  - Mobility and communication visualized if NetAnim is used.

**Experiment 2: Collecting and Analyzing Environmental Data**

- **Objective**: Collect and analyze environmental data from sensors using a microcontroller and visualize the data for further analysis.
- **Tools Required**:
    1. Microcontroller (e.g., Arduino or ESP32)
    2. DHT11 or DHT22 sensor (for temperature and humidity)
    3. USB cable
    4. Arduino IDE or PlatformIO
    5. Data visualization tool (e.g., Excel or Python Matplotlib)
- **Hardware Connections**:

    1. Connect the VCC pin of the DHT sensor to the 3.3V or 5V pin of the microcontroller.
    2. Connect the GND pin of the DHT sensor to the GND pin of the microcontroller.
    3. Connect the Data pin of the DHT sensor to a digital I/O pin on the microcontroller (e.g., D2).

- **Procedure**:

    1. Set up the hardware connections as per the schematic.
    2. Install the DHT library in the Arduino IDE (DHT sensor library by Adafruit).
    3. Write the program to read temperature and humidity data from the DHT sensor.
    4. Upload the program to the microcontroller.
    5. Open the Serial Monitor in the Arduino IDE to view the data in real time.
    6. Optionally, log the data to a file or visualize it using Python.

- **Arduino Code**:

```
#include <DHT.h>

#define DHTPIN 2 // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22 // DHT 22 (AM2302), DHT 11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");

  dht.begin();
}

void loop() {
  delay(2000); // Wait a few seconds between measurements

  // Reading temperature or humidity takes about 250 milliseconds!
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print values to the Serial Monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" *C");
}
```

- **Python Visualization Code**: Use Python to visualize the collected data by logging it to a file first and then using a tool like Matplotlib for plotting.

```
import matplotlib.pyplot as plt
import pandas as pd

# Load data from a CSV file
data = pd.read_csv("sensor_data.csv")

# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(data['Time'], data['Temperature'], label='Temperature (*C)')
plt.plot(data['Time'], data['Humidity'], label='Humidity (%)')
plt.xlabel('Time (s)')
plt.ylabel('Value')
plt.title('Temperature and Humidity Over Time')
plt.legend()
plt.grid()
plt.show()
```

- **Expected Outcome**:
    1. Real-time data displayed in the Serial Monitor for temperature and humidity.
    2. Visual representation of the data in a graph showing trends over time.
    3. Ability to analyze collected environmental data for patterns and insights

**Experiment 3: Measuring Energy Efficiency of Sensor Nodes**

- **Objective**: Measure the energy efficiency of sensor nodes in a wireless sensor network.
- **Tools Required**:
  1. NS3 Simulator
  2. Energy consumption model for sensor nodes
- **Procedure**:
  1. Install and set up NS3 simulator on your system.
  2. Create a wireless sensor network topology with multiple nodes using NS3.
  3. Install an energy model on each node to simulate battery-powered operation.
  4. Simulate data communication between the nodes.
  5. Analyze the energy consumed during the communication.
- **Sample Code (NS3 Energy Model Example)**:

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/wifi-module.h"
#include "ns3/energy-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-apps-module.h"

using namespace ns3;

int main(int argc, char *argv[]) {
  NodeContainer sensorNodes;
  sensorNodes.Create(10);

  // Configure WiFi
  WifiHelper wifi;
  wifi.SetStandard(WIFI_PHY_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
  wifiChannel.AddPropagationLoss("ns3::LogDistancePropagationLossModel");
  wifiPhy.SetChannel(wifiChannel.Create());

  WifiMacHelper wifiMac;
  wifiMac.SetType("ns3::AdhocWifiMac");

  NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, sensorNodes);

  // Install mobility model
  MobilityHelper mobility;
  mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                               "MinX", DoubleValue(0.0),
                               "MinY", DoubleValue(0.0),
                               "DeltaX", DoubleValue(5.0),
                               "DeltaY", DoubleValue(5.0),
                               "GridWidth", UintegerValue(5),
                               "LayoutType", StringValue("RowFirst"));
  mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
```

```
    mobility.Install(sensorNodes);

    // Install energy model
    BasicEnergySourceHelper energySourceHelper;
    energySourceHelper.Set("BasicEnergySourceInitialEnergyJ", DoubleValue(100.0));
    EnergySourceContainer energySources = energySourceHelper.Install(sensorNodes);

    WifiRadioEnergyModelHelper radioEnergyHelper;
    radioEnergyHelper.Set("TxCurrentA", DoubleValue(0.017));
    radioEnergyHelper.Set("RxCurrentA", DoubleValue(0.013));
    radioEnergyHelper.Install(devices, energySources);

    // Internet stack
    InternetStackHelper internet;
    internet.Install(sensorNodes);

    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = ipv4.Assign(devices);

    // Simulate data communication
    uint16_t port = 9;
    UdpEchoServerHelper echoServer(port);
    ApplicationContainer serverApps = echoServer.Install(sensorNodes.Get(0));
    serverApps.Start(Seconds(1.0));
    serverApps.Stop(Seconds(10.0));

    UdpEchoClientHelper echoClient(interfaces.GetAddress(0), port);
    echoClient.SetAttribute("MaxPackets", UintegerValue(5));
    echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    echoClient.SetAttribute("PacketSize", UintegerValue(1024));

    ApplicationContainer clientApps = echoClient.Install(sensorNodes.Get(9));
    clientApps.Start(Seconds(2.0));
    clientApps.Stop(Seconds(10.0));

    Simulator::Stop(Seconds(10.0));
    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

- **Expected Outcome**:
    1. Energy consumption data for each sensor node during the simulation.
    2. Insights into the energy efficiency of the nodes and network.
- **Result Analysis**: After running the simulation, observe the energy consumption statistics. Use the data to compare the efficiency of different communication protocols or node configurations.

**Experiment 4: Implementing a Basic Intrusion Detection System (IDS) in WSN**

- **Objective**: Implement a basic intrusion detection system (IDS) in a wireless sensor network to detect malicious activities or anomalies.
- **Tools Required**:
    1. NS3 Simulator
    2. Network topology with WSN nodes
    3. Intrusion detection logic or anomaly detection algorithm
- **Procedure**:
    1. Install and set up the NS3 simulator on your system.
    2. Create a WSN topology with multiple nodes using the NS3 simulator.
    3. Define normal network behavior, such as regular packet sizes and communication intervals.
    4. Implement an intrusion detection algorithm to monitor the network for abnormal activities (e.g., unusually large packets or frequent transmissions).
    5. Log intrusion events for analysis.
- **Sample Code (NS3 IDS Implementation)**:

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"

using namespace ns3;

// Function to simulate intrusion detection
bool DetectIntrusion (Ptr<Packet> packet) {
   if (packet->GetSize() > 1024) { // Example: Flag packets larger than 1024 bytes
      NS_LOG_UNCOND("Intrusion detected: Packet size exceeds threshold!");
      return true;
   }
   return false;
}

// Packet reception callback function
void ReceivePacket (Ptr<Socket> socket) {
   Ptr<Packet> packet = socket->Recv();
   if (DetectIntrusion(packet)) {
      NS_LOG_UNCOND("Intrusion logged for further analysis.");
   } else {
      NS_LOG_UNCOND("Normal packet received.");
   }
}

int main (int argc, char *argv[]) {
   // Create nodes
   NodeContainer nodes;
   nodes.Create(3);

   // Create a PointToPoint connection
   PointToPointHelper pointToPoint;
   pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
```

```
    pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));

    NetDeviceContainer devices = pointToPoint.Install(nodes);

    // Install Internet Stack
    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = address.Assign(devices);

    // Configure a UDP server
    uint16_t port = 9;
    UdpServerHelper server(port);
    ApplicationContainer serverApps = server.Install(nodes.Get(1));
    serverApps.Start(Seconds(1.0));
    serverApps.Stop(Seconds(10.0));

    // Configure a UDP client
    UdpClientHelper client(interfaces.GetAddress(1), port);
    client.SetAttribute("MaxPackets", UintegerValue(10));
    client.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    client.SetAttribute("PacketSize", UintegerValue(512));

    ApplicationContainer clientApps = client.Install(nodes.Get(0));
    clientApps.Start(Seconds(2.0));
    clientApps.Stop(Seconds(10.0));

    // Set up packet reception callback
    TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
    Ptr<Socket> recvSocket = Socket::CreateSocket(nodes.Get(1), tid);
    InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), port);
    recvSocket->Bind(local);
    recvSocket->SetRecvCallback(MakeCallback(&ReceivePacket));

    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

- **Expected Outcome**:
    1. Detection of intrusions based on predefined conditions (e.g., packet size threshold).
    2. Logging of intrusion events for further analysis.
    3. Normal packets processed without raising alerts.
- **Result Analysis**: Analyze the intrusion detection logs to evaluate the effectiveness of the implemented system. Check the accuracy of detection by varying the simulation parameters such as packet size and communication frequency.

## Experiment 5: Implementing and Evaluating Node Localization Algorithms in a WSN

• **Objective:** Implement a node localization algorithm in a wireless sensor network (WSN) and evaluate its performance in estimating the locations of sensor nodes.

• **Tools Required:**

1. NS3 Simulator or OMNeT++ Simulator
2. Localization algorithm (e.g., RSSI or Trilateration)
3. Network topology with sensor nodes

• **Procedure:**

1. Install and set up the NS3 or OMNeT++ simulator on your system.
2. Create a WSN topology with sensor nodes, where some nodes have known positions (anchors) and others have unknown positions.
3. Implement a localization algorithm such as trilateration or RSSI-based localization to estimate the positions of the unknown nodes.
4. Simulate the network and measure the localization accuracy by comparing the estimated positions with the actual positions.
5. Evaluate the performance of the algorithm by analyzing the localization error.

• **Sample Code (NS3 Trilateration Localization):**

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"

using namespace ns3;

// Function to estimate the position of a node using trilateration
Vector EstimatePosition (Vector anchor1, Vector anchor2, Vector anchor3, double d1, double
d2, double d3) {
   // Trilateration calculations here
   // This is just a placeholder for the actual algorithm
   double x = (d1 + d2 + d3) / 3.0;
   double y = (d1 + d2 + d3) / 3.0;
   return Vector(x, y, 0);
}

int main (int argc, char *argv[]) {
   // Setup and initialization code here
   // Example logic to use anchors and measure distances

   // Estimate the position of a node using the algorithm
   Vector estimatedPosition = EstimatePosition(Vector(0, 0, 0), Vector(100, 0, 0), Vector(50,
50, 0), 10.0, 10.0, 10.0);

   std::cout << "Estimated Position: " << estimatedPosition << std::endl;

   Simulator::Run();
```

```
    Simulator::Destroy();
    return 0;
}
```

• **Expected Outcome:**

1. Successful implementation of a node localization algorithm in a WSN.
2. Estimation of unknown node positions based on anchor nodes' known positions.
3. Evaluation of localization accuracy using error analysis.

• **Result Analysis:** Analyze the localization error, and compare the accuracy of the algorithm by varying simulation parameters such as the number of anchor nodes and transmission range.

## Experiment 6: Implementing a Simple IoT Communication Using the MQTT Protocol

• **Objective:** Implement a basic IoT communication system using the MQTT protocol to transfer data between sensor nodes and a central system.

• **Tools Required:**

1. MQTT Broker (e.g., Mosquitto)
2. IoT devices (sensor nodes or simulators)
3. Python or C++ for MQTT communication

• **Procedure:**

1. Install and set up the MQTT broker (Mosquitto) on your system.
2. Implement an MQTT publisher on a sensor node to send environmental data (e.g., temperature, humidity) to the broker.
3. Implement an MQTT subscriber on the central system to receive data from the sensor node.
4. Test the communication by sending data from the sensor and displaying it on the central system.

• **Sample Code (MQTT Publisher in Python):**

```
import paho.mqtt.client as mqtt
import random
import time

# Callback function when connected to broker
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("sensor/data")

# Callback function when a message is received
def on_message(client, userdata, msg):
    print(f"Received message: {msg.payload.decode()}")

# Setup MQTT client
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("localhost", 1883, 60)

# Publish random sensor data
while True:
    sensor_data = random.randint(20, 30)  # Example: Random temperature data
    client.publish("sensor/data", sensor_data)
    time.sleep(2)  # Publish every 2 seconds
```

• **Expected Outcome:**

1. Successful implementation of MQTT communication.
2. Data from a sensor node sent via MQTT to a central system.
3. Real-time data display on the subscriber side.

- **Result Analysis:** Evaluate the reliability of MQTT communication, and analyze the system's response to different data types, transmission delays, and message frequency.

## Experiment 7: Create a Simple Home Automation System Using IoT

• **Objective:** Design and implement a basic home automation system using IoT devices to control appliances such as lights, fans, or thermostats remotely.

• **Tools Required:**

1. IoT devices (e.g., Raspberry Pi, Arduino, sensors)
2. MQTT Protocol
3. Home automation platform (e.g., Home Assistant, Blynk)
4. Smart appliances (e.g., lights, fans)

• **Procedure:**

1. Set up IoT devices like Raspberry Pi or Arduino to control home appliances.
2. Connect the devices to a central platform (e.g., Blynk or Home Assistant).
3. Implement MQTT communication to send control signals (e.g., turning lights on/off).
4. Monitor and control the appliances remotely using a mobile or web application.

• **Sample Code (MQTT Control for Smart Light):**

```
import paho.mqtt.client as mqtt

# Setup MQTT client
client = mqtt.Client()
client.connect("localhost", 1883, 60)

# Publish command to turn on the light
client.publish("home/lights", "ON")

# Publish command to turn off the light
client.publish("home/lights", "OFF")
```

• **Expected Outcome:**

1. Successful integration of IoT devices for home automation.
2. Ability to control home appliances remotely using IoT platforms.
3. Real-time monitoring of appliance status.

• **Result Analysis:** Evaluate system performance and responsiveness. Analyze the impact of network delays, device reliability, and ease of use of the remote control interface.

**Experiment 8: Building a Basic IoT Environmental Monitoring System**

• **Objective:** Design and implement a simple IoT-based environmental monitoring system using sensors to monitor temperature, humidity, and air quality, with real-time data displayed on a central system.

• **Tools Required:**

1. Environmental sensors (temperature, humidity, air quality)
2. IoT platform (e.g., NodeMCU, Raspberry Pi)
3. MQTT Protocol
4. Centralized display system (e.g., web application)

• **Procedure:**

1. Set up environmental sensors to monitor temperature, humidity, and air quality.
2. Connect the sensors to an IoT device (e.g., NodeMCU or Raspberry Pi).
3. Use MQTT to transmit data from the sensors to a central server.
4. Display the sensor data on a web interface for real-time monitoring.

• **Sample Code (Environmental Monitoring with MQTT):**

```
import paho.mqtt.client as mqtt
import random
import time

# Setup MQTT client
client = mqtt.Client()
client.connect("localhost", 1883, 60)

# Simulate sensor readings and publish
while True:
    temperature = random.randint(20, 30)  # Example temperature data
    humidity = random.randint(30, 70)     # Example humidity data
    air_quality = random.randint(50, 100) # Example air quality data
    client.publish("environment/temperature", temperature)
    client.publish("environment/humidity", humidity)
    client.publish("environment/air_quality", air_quality)
    time.sleep(5)  # Publish every 5 seconds
```

• **Expected Outcome:**

1. Successful implementation of an IoT environmental monitoring system.
2. Real-time environmental data displayed on a central platform.
3. Monitoring of multiple environmental parameters (e.g., temperature, humidity, air quality).

• **Result Analysis:** Analyze the accuracy and consistency of the environmental data, and evaluate the reliability of the IoT system in terms of data transmission and user interaction.