Raspberry-Pi-Project

Table of Contents

1. NetSim Simulation of IoT Networks with Real-world Sensor Data from Raspberry Pi

Introduction

Project overview

- The project aims to integrate real-world sensor data from Raspberry Pi into an IoT network simulation of NetSim,enabling real-time monitoring and analysis.
- The key components of the project are the Raspberry Pi, four sensors (temperature, humidity, ultrasonic, and light-dependent sensor), and the NetSim simulator.
- The sensors will be connected to the Raspberry Pi using the GPIO pins. The data from the sensors will be collected and transmitted to the NetSim simulator.

Objective

- Demonstrate the seamless functionality of the integrated sensor network within the NetSim simulation environment.
- Evaluate the performance and reliability of the integrated sensor network under different scenarios.

Scope

- The project includes developing code to read data from real-world sensors (temperature, humidity, ultrasonic, and light-dependent sensor) connected to the Raspberry Pi.
- Analyzing the performance of the integrated sensor network under simulated scenarios and network conditions

Sensor Integration with Raspberry Pi

Sensor Description

In this section, we will discuss the sensors integrated with the Raspberry Pi and how their data is acquired and processed. The sensors included are:

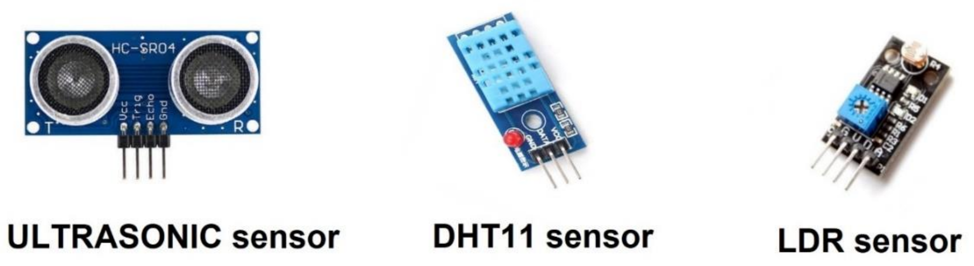


Figure 1-1: Different type of Sensors

a) Ultrasonic Sensor:

The ultrasonic sensor is a distance measuring device that uses sound waves to determine the distance between the sensor and an object.

Functionality: The ultrasonic sensor emits high-frequency sound pulses and measures the time taken for the echo to return after bouncing off an object.

Real-World Applications: Ultrasonic sensors are commonly used in robotics, automotive parking systems, industrial automation, and object detection applications.

b) DHT11 Sensor:

The DHT11 is a digital temperature and humidity sensor that provides reliable and low-cost sensing capabilities.

- The temperature sensor is designed to measure the ambient temperature of its surroundings. It provides accurate temperature readings, making it ideal for various environmental monitoring applications.

  Functionality: The temperature sensor operates by detecting changes in temperature, which cause variations in its electrical properties.

  Real-World Applications: The temperature sensor finds applications in weather monitoring, indoor climate control, industrial automation, and more.

- The humidity sensor, also known as a hygrometer, measures the moisture content or relative humidity in the atmosphere.

  Functionality: The humidity sensor functions by sensing the changes in the moisture level, which affects its electrical characteristics.

  Real-World Applications: Humidity sensors are widely used in weather stations, greenhouses, HVAC systems, and industries where humidity control is critical.

c) LDR Sensor:

The light-dependent sensor, also known as an LDR (Light-Dependent Resistor) or photoresistor, detects changes in ambient light intensity.

Functionality: The LDR's resistance changes with variations in light intensity, allowing it to detect the presence or absence of light.

Real-World Applications: Light-dependent sensors are used in streetlights, security systems, camera exposure control, and automatic lighting systems.

Wiring and Connection to Raspberry Pi

All sensors (Ultrasonic, DHT11, and LDR) are connected to the Raspberry Pi using specific GPIO pins as follows:

a. **Ultrasonic Sensor**

   VCC Pin connected to 5v (Physical Pin 2)

   TRIG Pin connected to GPIO 17 (Physical Pin 11)

   ECHO Pin connected to GPIO 27 (Physical Pin 13)

   GND Pin connected GND (Physical Pin 6)

b. **DHT11 Sensor (Temperature and Humidity)**

   VCC Pin connected to 5v (Physical Pin 4)

   DATA Pin connected to GPIO 22 (Physical Pin 15)

   GND Pin connected GND (Physical Pin 20)

c. **Light-Dependent Resistor (LDR):**

   VCC Pin connected to 3v3 (Physical Pin 1)

   DATA Pin connected to GPIO 04 (Physical Pin 7)

   GND Pin connected GND (Physical Pin 39)

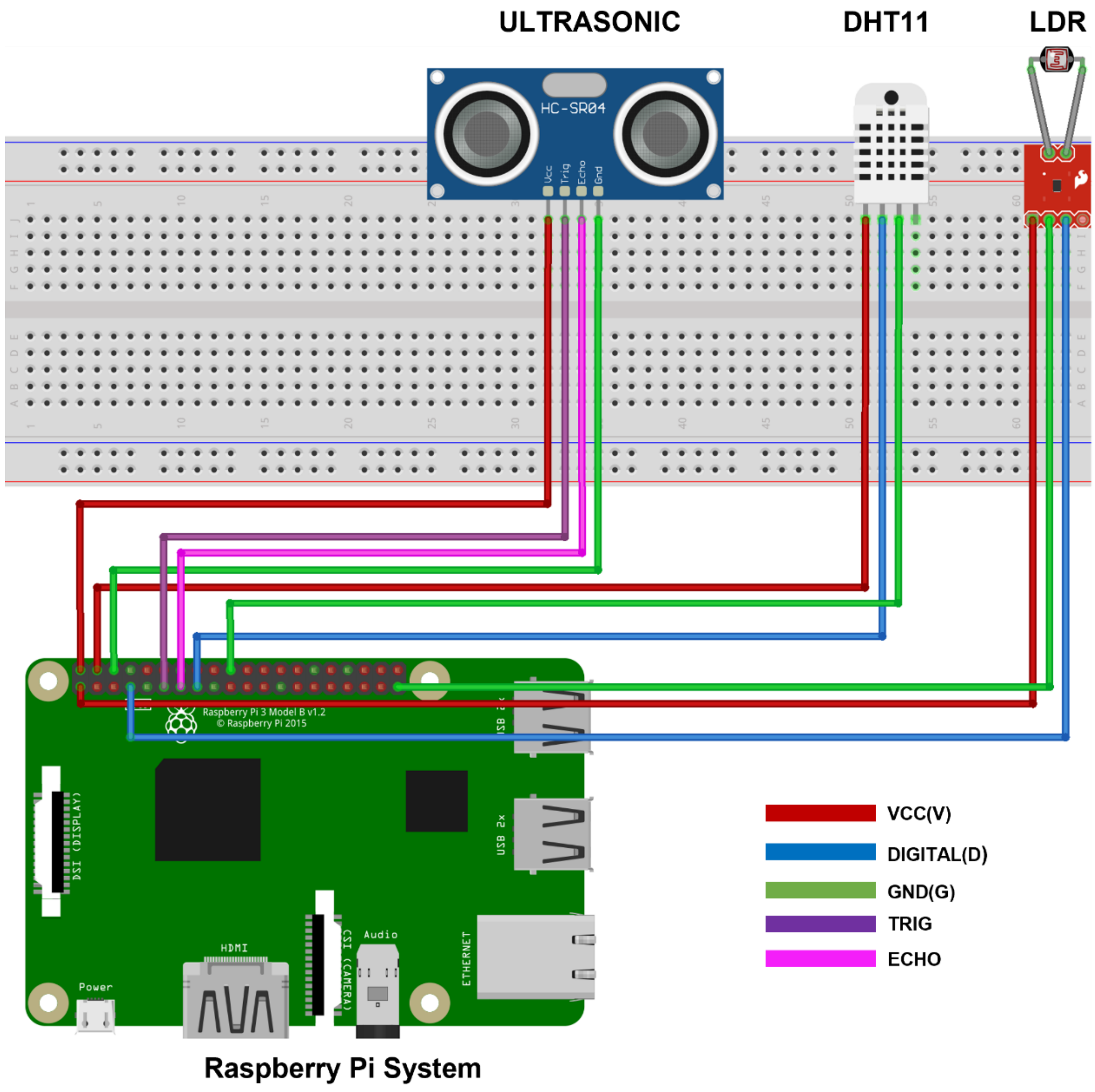


Figure 1-2: Connection Diagram of Sensors

Communication Protocols Used

a. Ultrasonic Sensor

GPIO (General Purpose Input/Output) pins are used to communicate with the Ultrasonic sensor.One GPIO pin triggers the sensor, and another GPIO pin receives the echo signal.Distance to an object is calculated based on the time taken for the echo signal to return.

b. DHT11 Sensor (Temperature and Humidity)

The DHT11 sensor communicates using a one-wire digital interface via a single GPIO pin.It sends temperature and humidity data to the Raspberry Pi through the data pin.

c. Light-Dependent Resistor (LDR):

The LDR sensor is connected to a digital input pin of the Raspberry Pi GPIOinterface.The sensor provides a digital output that varies with light intensity.The digital output is directly read by the Raspberry Pi, and its value corresponds to the light intensity detected by the sensor

How sensor data is acquired and processed by the Raspberry Pi

Data Acquisition:

- Ultrasonic Sensor: The Raspberry Pi triggers the sensor and measures the time taken for the echo to return. It calculates the distance based on the speed of sound in air.
- DHT11 Sensor: The Raspberry Pi reads digital data from the sensor using a specific protocol, extracting humidity and temperature values.
- LDR Sensor: The Raspberry Pi reads the digital output from the sensor to obtain the light intensity value.

Data Processing:

- Ultrasonic Sensor: No additional processing is required for the distance data.
- DHT11 Sensor: The raw data is processed to obtain humidity and temperature values.
- LDR Sensor: No significant data processing is applied to the light intensity readings.

Data Transmission:

- The Raspberry Pi communicates with the NetSim simulator using socket programming and sends the sensor data as packets over the network.

Server Implementation

Explanation of server.c code

The server.c code serves as the central component for collecting data from the sensors connected to the Raspberry Pi and transmitting this data to the client, which is the NetSim simulator. The code uses the WiringPi library for GPIO interfacing.

Note*: WiringPi is a C/C++ library that provides an easy-to-use interface to the GPIO pins on the Raspberry Pi. It was originally developed by Gordon Henderson, and it is now maintained by a team of developers.*

Inside the main loop, the server reads data from each sensor and forms a response containing temperature, humidity, ultrasonic distance, and light intensity values. Before sending the data, it performs a validity check to ensure the sensor readings are within the expected range.

In summary, server.c serves as a bridge between the physical sensors on the Raspberry Pi and the NetSim simulator, ensuring the real-world sensor data is integrated into the simulated IoT network environment.

Note*: The entire server.c code will be provided in the appendix at the end of the document*.

How the server collects and handles data from the sensors

The server collects data from the connected sensors on the Raspberry Pi and sends it to the NetSim simulator through a socket connection. It ensures that the data is valid and includes temperature, humidity, ultrasonic distance, and light intensity values

Communication between Raspberry Pi and the NetSim Simulator

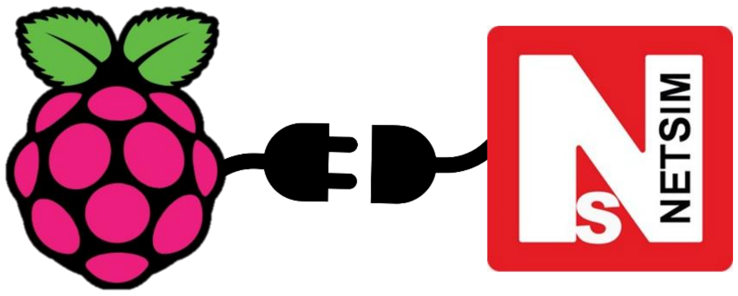


Figure 1-3: Communication between Raspberry Pi and NetSim

The communication between the Raspberry Pi and the NetSim simulator is established through a socket connection. The server.c code running on the Raspberry Pi acts as the server, while the NetSim simulator's client.c code acts as the client.

- **Server**: The server.c code initializes a socket and binds it to a specific IP address and port on the Raspberry Pi. It listens for incoming connections from the NetSim client.
- **Client**: The NetSim client.c code initializes a socket and connects it to the server's IP address and port.
- **Data Transmission**: Once the connection is established, the server collects sensor data from the Raspberry Pi and sends it to the NetSim client. The client receives the data and processes it for simulation within the NetSim environment.
- **Real-time Integration**: The real-world sensor data from the Raspberry Pi is now integrated seamlessly into the virtual IoT network simulation in NetSim, enabling real-time interactions between the virtual sensor nodes and the server

This communication mechanism ensures that the simulated IoT network in NetSim receives and processes real-world sensor data from the Raspberry Pi, creating a more realistic and practical simulation environment.

NetSim Simulator Configuration

Overview of NetSim Simulator and its features

NetSim is a network simulator tool enables users to virtually create a network comprising of devices, links, applications etc.that can be used to model and simulate different types of networks, including wireless networks, wired networks, and sensor networks. and study the behavior and performance of the Network.

NetSim comes with a wide range of features:

- **Comprehensive Technology Coverage**: NetSim supports a wide range of network technologies, including internetworks, legacy, and cellular networks, mobile adhoc networks, software-defined networks, wireless sensor networks, internet of things, cognitive radio networks, LTE networks, 5G NR (Pro version), VANET (Pro version), and more.

- **Performance Reporting:** The simulator provides detailed performance metrics for networks and sub-networks, enabling users to analyze and evaluate network behavior.
- **Powerful Visualization Tools:** NetSim includes a packet animator, allowing users to visualize the flow of packets in the simulated network, making it easier to understand the data transmission process.
- **External Interfacing:** NetSim offers external interfacing with SUMO, MATLAB, and Wireshark, providing flexibility in data analysis and integration with other tools.
- **Integrated Debugging (Standard and Pro versions):** Users can write and debug their custom code, linking it to NetSim using Visual Studio, facilitating advanced research and experimentation.

Client.c code in the NetSim simulator

In the NetSim simulator **Client.c** file is added to the Application project

The "client.c" code serves as a client application responsible for connecting to a server and receiving data from it. It uses the Winsock library for socket communication on Windows systems.

- **Initialization:** The code initializes Winsock, creates a client socket, and sets up the server's address
- **Data Reception:**The client continuously reads data from the server using the "recv" function and stores it in the "buffer" variable.
- **Connection Closing:** After data reception is complete, the client closes the socket and cleans up Winsock resources.

Note: *The entire client.c code will be provided in the appendix at the end of the document*.

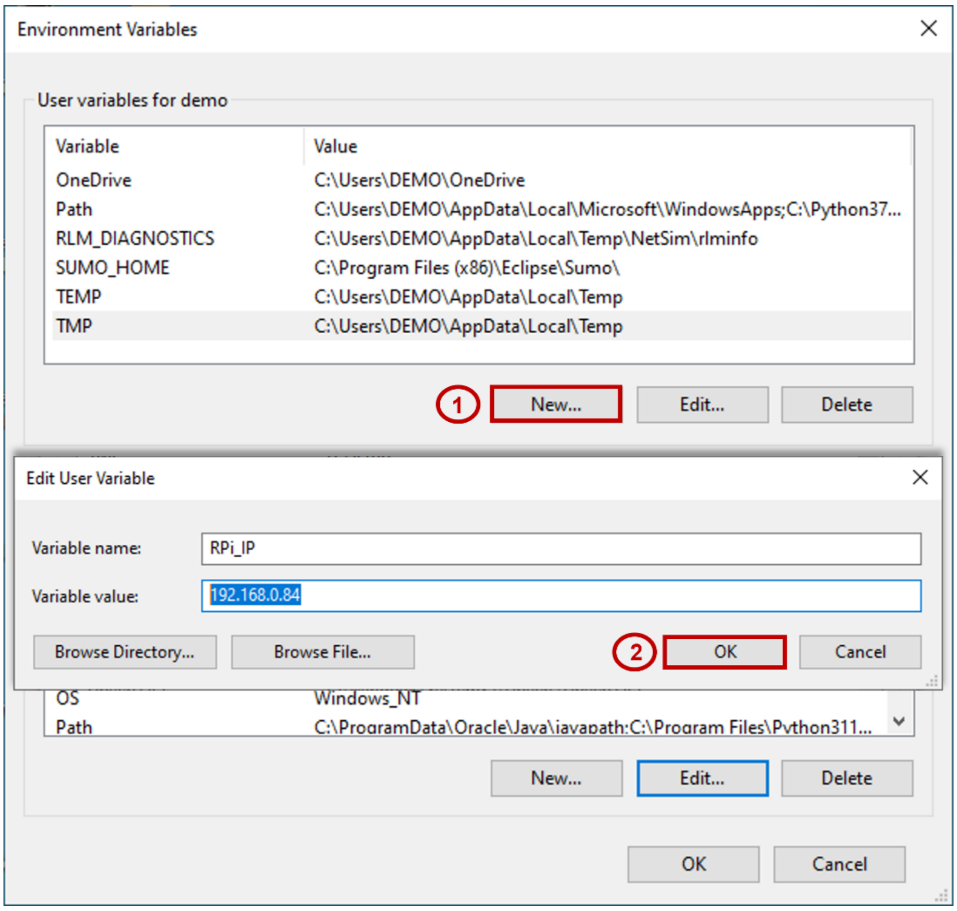
Environment variable Configuration

Raspberry Pi Server IP will be added in Environment Variables of windows system were we are Running NetSim Simulator

1. Go to start search Run → Enter the command “SystemPropertiesAdvanced” and then click on OK.

2. Click the Environment Variables → Add the following Variables.

Variable name - RPi-IP
Variable value - 192.168.0.84 **(IP-Address of your Rapberry Pi system)**


Environment Variables

User variables for demo

Variable	Value
OneDrive	C:\Users\DEMO\OneDrive
Path	C:\Users\DEMO\AppData\Local\Microsoft\WindowsApps;C:\Python37...
RLM_DIAGNOSTICS	C:\Users\DEMO\AppData\Local\Temp\NetSim\rlminfo
SUMO_HOME	C:\Program Files (x86)\Eclipse\Sumo\
TEMP	C:\Users\DEMO\AppData\Local\Temp
TMP	C:\Users\DEMO\AppData\Local\Temp

Edit User Variable

Variable name: RPi_IP

Variable value: 192.168.0.84

OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\Program Files\Python311...

Figure 1-4: Environmental Variable settings

Creating and Running Sensor Network Scenario in NetSim

1. The NetSim-Raspbery-Pi-Interfacing-v13.3 comes with a sample network configuration that are already saved. To open this example, Go to Your work in the home screen of NetSim and click on the Experiment.
2. The saved network scenario consists
    - 4 Sensor Nodes(Four different sensors)
    - 1 LOWPAN_Gateway
    - 1 Router
    - 1 Wired-Node(Server)

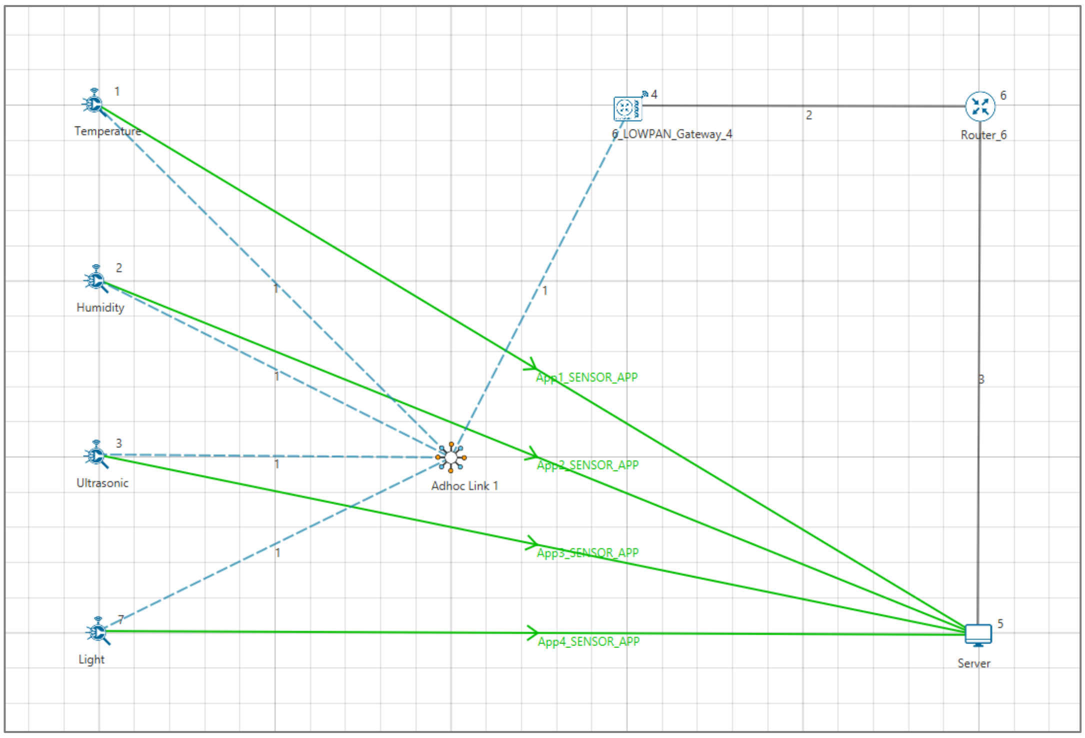


Figure 1-5: Virtual network scenario created in NetSim with real sensor data from Raspberry Pi.

Note: *This network scenario is created in Internet of things Network in NetSim*

3. Application Properties Configuration

Application Properties	
Application Type	SENSOR
Source ID	1,2,3,4(Four different Sensor ID,s)
Destination ID	5 (Server ID)

Table 1-Application Properties

4. Before running the NetSim scenario please assure that server code is running in Raspberry-Pi system

Example to run Server code on raspbery pi system:

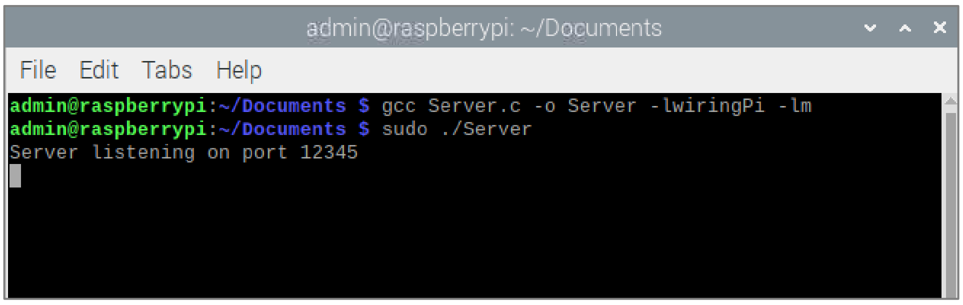
- Save Server.c code on Documents
- Open the terminal from the Documents

Give the follwing commands:-

```
gcc<filename.c>-o<filename> -lwiringPi -lm

sudo ./filename
```

Note: *Here filename.c is the name of your server.c code*

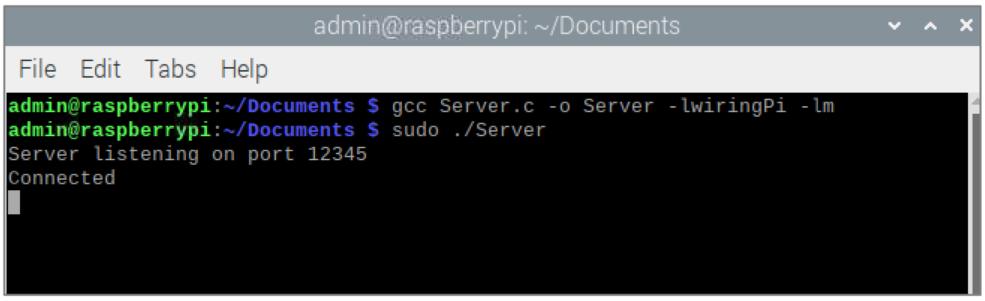


```
admin@raspberrypi:~/Documents $ gcc Server.c -o Server -lwiringPi -lm
admin@raspberrypi:~/Documents $ sudo ./Server
Server listening on port 12345
```

Figure 1-6: Running Server.c code on Raspberry pi system

Once you started the Server.c code it starts listening client to connect

5. After starting the server, you can run the NetSim example. As the scenario begins running, you can go back to the Raspberry Pi system, where you will notice that the server status changes to 'Connected



```
admin@raspberrypi:~/Documents $ gcc Server.c -o Server -lwiringPi -lm
admin@raspberrypi:~/Documents $ sudo ./Server
Server listening on port 12345
Connected
```

Figure 1-7: After Client is connected to server

Results and Discussions

After the simulation is completed, we can check the results using Wireshark captured files. In the Result Dashboard, On the left side, Packet Capture →Simulation→We can see all captured packects of all different sensors

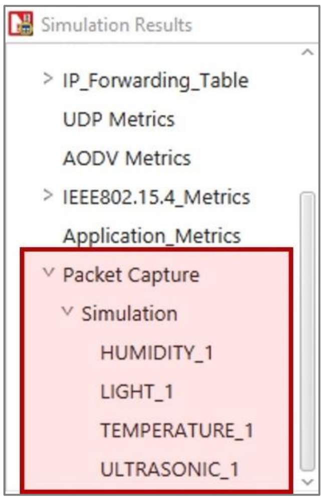
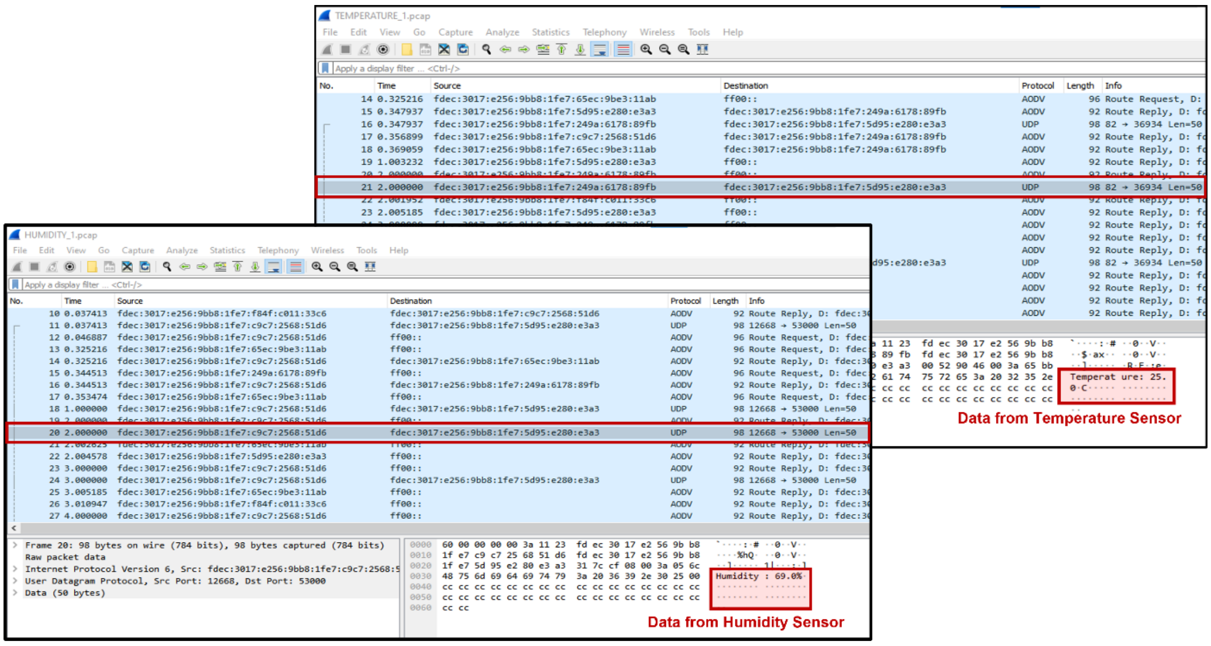


Figure 1-8: Packet capture outputs in Result Dashboard

We can access the pcap files for each individual sensor. These pcap files contain the real payload data transmitted during the simulation, allowing us to examine and interpret the sensor data in detail.



TEMPERATURE_1.pcap

```
No.  Time      Source                                  Destination                             Protocol Length Info
  14 0.325216  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       96 Route Request, D:
  15 0.347937  fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 fdec:3017:e256:9bb8:1fe7:249a:6178:89fb AODV       92 Route Reply, D: fd
  16 0.347937  fdec:3017:e256:9bb8:1fe7:249a:6178:89fb fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 82 → 36934 Len=50
  17 0.356899  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:249a:6178:89fb AODV       92 Route Reply, D: fd
  18 0.369059  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab fdec:3017:e256:9bb8:1fe7:249a:6178:89fb AODV       92 Route Reply, D: fd
  19 1.003232  fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 ff00::                                  AODV       92 Route Reply, D: fd
  21 2.000000  fdec:3017:e256:9bb8:1fe7:249a:6178:89fb fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 82 → 36934 Len=50
  23 2.005185  fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 ff00::                                  AODV       92 Route Reply, D: fd
```

```
a 11 23  fd ec 30 17 e2 56 9b b8   `····:·# ··0··V··
3 89 fb  fd ec 30 17 e2 56 9b b8   ··$·ax·· ··0··V··
0 e3 a3  00 52 90 46 00 3a 65 bb
2 61 74  75 72 65 3a 20 32 35 2e   Temperat ure: 25.
c cc cc  cc cc cc cc cc cc cc cc   0·C····· ········
c cc cc  cc cc cc cc cc cc cc cc
```

Data from Temperature Sensor

HUMIDITY_1.pcap

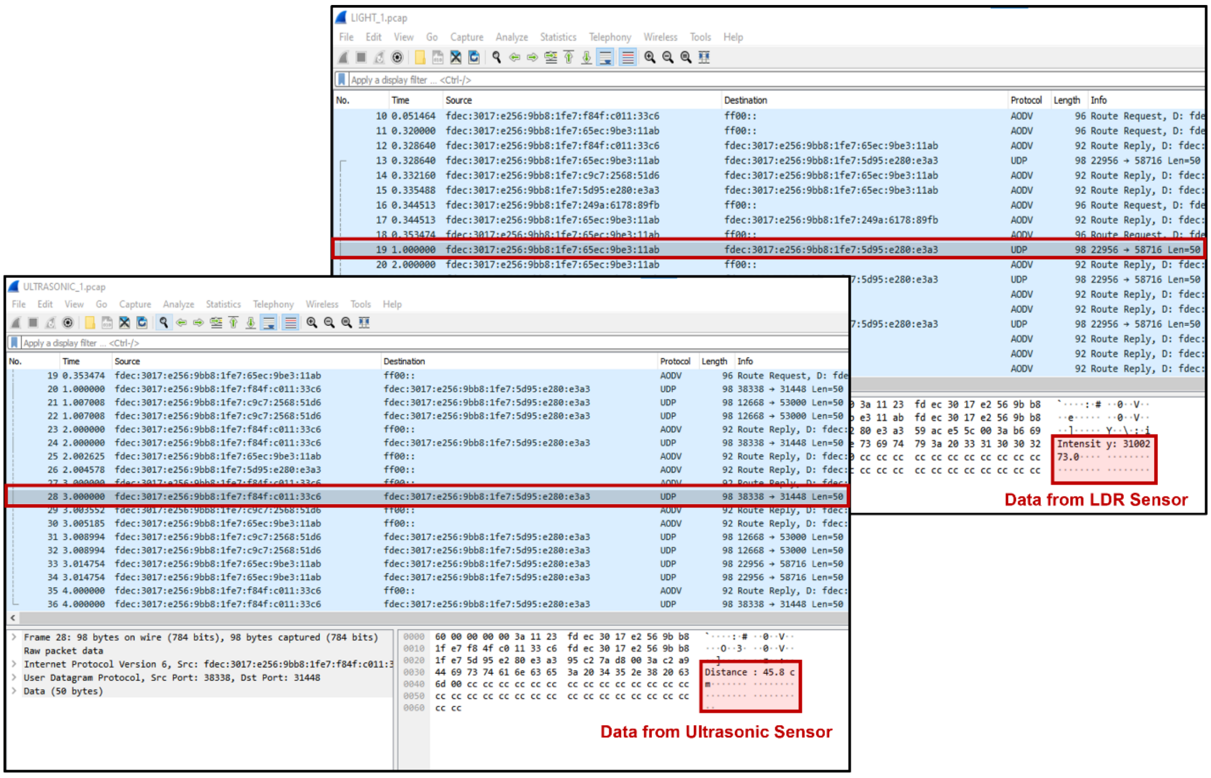
```
No.  Time      Source                                  Destination                             Protocol Length Info
  10 0.037413  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 AODV       92 Route Reply, D: fdec:3
  11 0.037413  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  12 0.046887  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 ff00::                                  AODV       96 Route Request, D: fdec
  13 0.325216  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       96 Route Request, D: fdec
  14 0.325216  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab AODV       92 Route Reply, D: fdec:3
  15 0.344513  fdec:3017:e256:9bb8:1fe7:249a:6178:89fb ff00::                                  AODV       96 Route Request, D: fdec
  16 0.344513  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:249a:6178:89fb AODV       92 Route Reply, D: fdec:3
  17 0.353474  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       96 Route Request, D: fdec
  18 1.000000  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  20 2.000000  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  22 2.004578  fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 ff00::                                  AODV       92 Route Reply, D: fdec:3
  23 3.000000  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 ff00::                                  AODV       92 Route Reply, D: fdec:3
  24 3.000000  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  25 3.005185  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       92 Route Reply, D: fdec:3
  26 3.010947  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 ff00::                                  AODV       92 Route Reply, D: fdec:3
  27 4.000000  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 ff00::                                  AODV       92 Route Reply, D: fdec:3
```

```
> Frame 20: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
  Raw packet data
> Internet Protocol Version 6, Src: fdec:3017:e256:9bb8:1fe7:c9c7:2568:5
> User Datagram Protocol, Src Port: 12668, Dst Port: 53000
> Data (50 bytes)
```

```
0000  60 00 00 00 00 3a 11 23  fd ec 30 17 e2 56 9b b8   `····:·# ··0··V··
0010  1f e7 c9 c7 25 68 51 d6  fd ec 30 17 e2 56 9b b8   ····%hQ· ··0··V··
0020  1f e7 5d 95 e2 80 e3 a3  31 7c cf 08 00 3a 05 6c   ··]····· 1|···:·l
0030  48 75 6d 69 64 69 74 79  3a 20 36 39 2e 30 25 00   Humidity : 69.0%·
0040  cc cc cc cc cc cc cc cc  cc cc cc cc cc cc cc cc   ········ ········
0050  cc cc cc cc cc cc cc cc  cc cc cc cc cc cc cc cc   ········ ········
0060  cc cc
```

Data from Humidity Sensor

Figure 1-9: Temperature and Humidity sensor data



LIGHT_1.pcap

```
No.  Time      Source                                  Destination                             Protocol Length Info
  10 0.051464  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 ff00::                                  AODV       96 Route Request, D: fde
  11 0.320000  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       96 Route Request, D: fde
  12 0.328640  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab AODV       92 Route Reply, D: fdec:
  13 0.328640  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 22956 → 58716 Len=50
  14 0.332160  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab AODV       92 Route Reply, D: fdec:
  15 0.335488  fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab AODV       92 Route Reply, D: fdec:
  16 0.344513  fdec:3017:e256:9bb8:1fe7:249a:6178:89fb ff00::                                  AODV       96 Route Request, D: fde
  17 0.344513  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab fdec:3017:e256:9bb8:1fe7:249a:6178:89fb AODV       92 Route Reply, D: fdec:
  19 1.000000  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 22956 → 58716 Len=50
  20 2.000000  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       92 Route Reply, D: fdec:
```

```
0 3a 11 23  fd ec 30 17 e2 56 9b b8   `····:·# ··0··V··
b e3 11 ab  fd ec 30 17 e2 56 9b b8   ··e····· ··0··V··
2 80 e3 a3  59 ac e5 5c 00 3a b6 69   ··]····· Y··\·:·i
e 73 69 74  79 3a 20 33 31 30 30 32   Intensit y: 31002
0 cc cc cc  cc cc cc cc cc cc cc cc   73.0···· ········
c cc cc cc  cc cc cc cc cc cc cc cc   ········ ········
```

Data from LDR Sensor

ULTRASONIC_1.pcap

```
No.  Time      Source                                  Destination                             Protocol Length Info
  19 0.353474  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       96 Route Request, D: fde
  20 1.000000  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 38338 → 31448 Len=50
  21 1.007008  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  22 1.007008  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  23 2.000000  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 ff00::                                  AODV       92 Route Reply, D: fdec:
  24 2.000000  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 38338 → 31448 Len=50
  25 2.002625  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       92 Route Reply, D: fdec:
  26 2.004578  fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 ff00::                                  AODV       92 Route Reply, D: fdec:
  28 3.000000  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 38338 → 31448 Len=50
  30 3.005185  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab ff00::                                  AODV       92 Route Reply, D: fdec:
  31 3.008994  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  32 3.008994  fdec:3017:e256:9bb8:1fe7:c9c7:2568:51d6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 12668 → 53000 Len=50
  33 3.014754  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 22956 → 58716 Len=50
  34 3.014754  fdec:3017:e256:9bb8:1fe7:65ec:9be3:11ab fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 22956 → 58716 Len=50
  35 4.000000  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 ff00::                                  AODV       92 Route Reply, D: fdec:
  36 4.000000  fdec:3017:e256:9bb8:1fe7:f84f:c011:33c6 fdec:3017:e256:9bb8:1fe7:5d95:e280:e3a3 UDP        98 38338 → 31448 Len=50
```

```
> Frame 28: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
  Raw packet data
> Internet Protocol Version 6, Src: fdec:3017:e256:9bb8:1fe7:f84f:c011:3
> User Datagram Protocol, Src Port: 38338, Dst Port: 31448
> Data (50 bytes)
```

```
0000  60 00 00 00 00 3a 11 23  fd ec 30 17 e2 56 9b b8   `····:·# ··0··V··
0010  1f e7 f8 4f c0 11 33 c6  fd ec 30 17 e2 56 9b b8   ···O··3· ··0··V··
0020  1f e7 5d 95 e2 80 e3 a3  95 c2 7a d8 00 3a c2 a9   ··]····· ··z··:··
0030  44 69 73 74 61 6e 63 65  3a 20 34 35 2e 38 20 63   Distance : 45.8 c
0040  6d 00 cc cc cc cc cc cc  cc cc cc cc cc cc cc cc   m······· ········
0050  cc cc cc cc cc cc cc cc  cc cc cc cc cc cc cc cc   ········ ········
0060  cc cc                                              ··
```

Data from Ultrasonic Sensor

Figure 1-10: Ultrasonic and LDR sensor data

In the provided above Wireshark capture file screenshots, the payload data contains information about temperature,humidity,Ultrasonic and LDR sensor readings. These readings are the actual values acquired from the physical sensors connected to the Raspberry Pi. By examining these captured packets, we can verify the successful transmission of sensor data from the Raspberry Pi server to the NetSim simulator

Appendix: NetSim source code modifications

MS Visual Studio Development environment is required for editing and building NetSim source codes. Please see this link on setting up Visual Studio

https://support.tetcos.com/support/solutions/articles/14000138721-what-components-ofvisual-studio-community-2022-to-install-and-configure-to-work-with-netsim-source-c

To open our project source code section, in NetSim home screen to →your work → source code → open code.

NetSim comes with inbuilt low-level functions to capture packets. This code is not open for user modification. The code to access the payload/header and to modify the payload/header is open to users and can be modified. We show below the source code changes we have made in red.

Server.c code at the Raspberry pi system

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <wiringPi.h>

#define PORT 12345
#define LDR_PIN 7       // Physical Pin 7
#define TRIG_PIN 0      // Physical Pin 11 (GPIO 17)
#define ECHO_PIN 2      // Physical Pin 13 (GPIO 27)
#define DHT_PIN 3       // Physical Pin 15 (GPIO 22)
#define DHT_TYPE DHT11  // DHT11 Sensor

int readLDRValue()
{
   pinMode(LDR_PIN, INPUT);
```

```
    int pulse_count = 0;
    int start_time = millis();

    while ((millis() - start_time) < 1000)  // Count pulses for 1 second
    {
        if (digitalRead(LDR_PIN) == LOW)
        {
            pulse_count++;
        }
    }

    return pulse_count;  // Use pulse count as an estimate of light intensity
}

float readUltrasonicDistance()
{
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    digitalWrite(TRIG_PIN, LOW);
    delay(30);

    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(20);
    digitalWrite(TRIG_PIN, LOW);

    while (digitalRead(ECHO_PIN) == LOW)
    {
        continue;
    }

    long startTime = micros();
    while (digitalRead(ECHO_PIN) == HIGH)
    {
        continue;
```

```
  }

  long travelTime = micros() - startTime;
  float distance = travelTime / 58.0;  // Speed of sound in air is approximately 340 m/s or 29
microseconds per centimeter
  return distance;
}

void readDHT11Data(float *humidity, float *temperature)
{
  int data[5] = {0, 0, 0, 0, 0};

  pinMode(DHT_PIN, OUTPUT);
  digitalWrite(DHT_PIN, LOW);
  delay(18);
  digitalWrite(DHT_PIN, HIGH);
  delayMicroseconds(40);
  pinMode(DHT_PIN, INPUT);

  // Wait for sensor response
  while (digitalRead(DHT_PIN) == HIGH)
    delayMicroseconds(1);

  // Sensor should pull low for 80us
  while (digitalRead(DHT_PIN) == LOW)
    delayMicroseconds(1);

  // Sensor should pull high for 80us
  while (digitalRead(DHT_PIN) == HIGH)
    delayMicroseconds(1);

  // Read 40 bits of data
  for (int i = 0; i < 40; i++)
  {
    // Data bit starts with a low state for 50us
    while (digitalRead(DHT_PIN) == LOW)
        delayMicroseconds(1);

    // Measure the length of the high state to determine if it's a 0 or 1
    int counter = 0;
    while (digitalRead(DHT_PIN) == HIGH)
    {
        delayMicroseconds(1);
        counter++;
    }

    // Store the bit by shifting data array
    data[i / 8] <<= 1;
    if (counter > 30)
        data[i / 8] |= 1;
  }

  // Verify checksum
  if (data[4] == ((data[0] + data[1] + data[2] + data[3]) & 0xFF))
  {
    *humidity = (float)data[0];
    *temperature = (float)data[2];
  }
  else
  {
    *humidity = -1.0;    // Error value for humidity
    *temperature = -1.0; // Error value for temperature
  }
}

int main()
{
  if (wiringPiSetup() == -1)
```

```
{
    printf("Failed to initialize WiringPi library.\n");
    return 1;
}

int server_fd, new_socket, valread;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[1024] = {0};
char response[1024] = {0};

// Create socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Bind the socket to given IP and port
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_fd, 3) < 0)
{
    perror("listen failed");
    exit(EXIT_FAILURE);
    }

    printf("Server listening on port %d\n", PORT);

    // Accept incoming connection
  if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0)
    {
        perror("accept failed");
        exit(EXIT_FAILURE);
    }

    printf("Connected\n");

    while (1)
    {
        // Read LDR sensor data and calculate light intensity
        int ldrValue = readLDRValue();

        // Read Ultrasonic sensor data
        float ultrasonicDistance = readUltrasonicDistance();

        // Read DHT11 sensor data
        float humidity, temperature;
        readDHT11Data(&humidity, &temperature);

        // Send the sensor data to the client

        if (humidity != -1.0 && temperature != -1.0 && ldrValue!= -1.0 && ultrasonicDistance>0)
        {
            sprintf(response, "\nTemperature: %.1f°C, Humidity: %.1f%% Ultrasonic Distance:
%.2f cm light Intensity: %d ", temperature, humidity,ultrasonicDistance,ldrValue);
        }
        else
        {
            sprintf(response, "Failed to retrieve data from sensor.\n");
        }
        send(new_socket, response, strlen(response), 0);


        delay(1000);
    }


    close(new_socket);
    close(server_fd);


    return 0;
}
```

Client.c code is file added to the Application Project in NetSim

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#include "Application.h"

#pragma comment(lib, "ws2_32.lib")
#define PORT 12345
char* SERVER_IP;
WSADATA wsaData;
SOCKET sock;
struct sockaddr_in server;
char buffer[1024] = { 0 };

void Client_init()
{
    SERVER_IP= getenv("RPi_IP");

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        fprintf(stderr, "Failed to initialize Winsock\n");
        return 1;
    }

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        fprintf(stderr, "Failed to create socket\n");
        WSACleanup();
        return 1;
    }

    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);

    // Connect to server
    if (connect(sock, (struct sockaddr*)&server, sizeof(server)) == SOCKET_ERROR)
    {
        fprintf(stderr, "Connection failed\n");
        closesocket(sock);
        WSACleanup();
        return 1;
    }
}
char* client_run()
{

    // Receive and print data from server
    while (1)
    {
        memset(buffer, 0, sizeof(buffer));
        int bytesRead = recv(sock, buffer, sizeof(buffer) - 1, 0);
        if (bytesRead <= 0)
            break;
        //fprintf(stderr, "%s", buffer);
        return buffer;
    }
}
void client_end()
{
    closesocket(sock);
    WSACleanup();
}
```

Changes in **copy_payload() in Application.c in Application Project in NetSim**

```
void copy_payload(UINT8 real[],NetSim_PACKET* packet,unsigned int* payload, ptrAPPLICATION_INFO info)
{
    u_short i;
    uint32_t key = 16;
    char* buffer= client_run();
    char temp_val[BUFSIZ];
    char humidi_val[BUFSIZ];
    char distan_val[BUFSIZ];
    char light_val[BUFSIZ];
   if (payload)
   {
          if (buffer)
          {
              char* DNAME = DEVICE_NAME(pstruEventDetails->nDeviceId);
```

```
      //STart

          char* tempStart = strstr(buffer, "Temperature: ");
          char* humidityStart = strstr(buffer, "Humidity: ");
          char* distanceStart = strstr(buffer, "Ultrasonic Distance: ");
          char* lightStart = strstr(buffer, "light Intensity: ");

          if (strcmp(DNAME, "TEMPERATURE") == 0) {
              if (tempStart != NULL) {
                  tempStart += strlen("Temperature: ");
                  double temperature;
                  sscanf(tempStart, "%lf%%", &temperature);
                  //printf("Temperature: %.1lf°C", temperature);
                  sprintf(temp_val, "Temperature: %.1lf°C", temperature);
                  for (i = 0; i < sizeof(temp_val); i++)
                {
                        if (info->encryption == Encryption_XOR)
                            real[i] = xor_encrypt(temp_val[i], 16);
                         else
                            real[i] = temp_val[i];
                }
        }
}

      else if (strcmp(DNAME, "HUMIDITY") == 0) {
            if (humidityStart != NULL) {
                humidityStart += strlen("Humidity: ");
                double humidity;
                sscanf(humidityStart, "%lf%%", &humidity);
                //printf();
               sprintf(humidi_val, "Humidity: %.1lf%%", humidity);
               for (i = 0; i < sizeof(humidi_val); i++)
                 {
                       if (info->encryption == Encryption_XOR)
```

```
                    real[i] = xor_encrypt(humidi_val[i], 16);
                  else
                     real[i] = humidi_val[i];
                }
         }
  }
 else if (strcmp(DNAME, "ULTRASONIC") == 0) {
         if (distanceStart != NULL) {
             distanceStart += strlen("Ultrasonic Distance: ");
             double ultrasonicDistance;
             sscanf(distanceStart, "%lf cm", &ultrasonicDistance);
             //printf();
             sprintf(distan_val, "Distance: %.1lf cm", ultrasonicDistance);
             for (i = 0; i < sizeof(distan_val); i++)
             {
             if (info->encryption == Encryption_XOR)
                 real[i] = xor_encrypt(distan_val[i], 16);
             else
                real[i] = distan_val[i];
               }
       }
}
   else if (strcmp(DNAME, "LIGHT") == 0) {
             if (lightStart != NULL) {
             lightStart += strlen("light Intensity: ");
             double lightIntensity;
             sscanf(lightStart, "%lf", &lightIntensity);
             //printf();
              sprintf(light_val, "Intensity: %.1lf", lightIntensity);
              for (i = 0; i < sizeof(light_val); i++)
              {
             if (info->encryption == Encryption_XOR)
                 real[i] = xor_encrypt(light_val[i], 16);
              else
```

```
                                real[i] = light_val[i];
                            }
                    }
            }
        }
        if (info->encryption == Encryption_TEA)
         encryptBlock(real, payload, &key);
        else if (info->encryption == Encryption_AES)
            aes256(real,payload);
        else if(info->encryption==Encryption_DES)
            des(real,payload);
        }
}
```

```
void copy_payload(UINT8 real[],NetSim_PACKET* packet,unsigned int* payload, ptrAPPLICATION_INFO info)
{
	u_short i;
	uint32_t key = 16;
	 char* buffer= client_run();
	 char temp_val[BUFSIZ];
	 char humidi_val[BUFSIZ];
	 char distan_val[BUFSIZ];
	 char light_val[BUFSIZ];
	if (payload)
	{
		if (buffer)
		{
			char* DNAME = DEVICE_NAME(pstruEventDetails->nDeviceId);

			//STart

				char* tempStart = strstr(buffer, "Temperature: ");
				char* humidityStart = strstr(buffer, "Humidity: ");
				char* distanceStart = strstr(buffer, "Ultrasonic Distance: ");
				char* lightStart = strstr(buffer, "light Intensity: ");

				if (strcmp(DNAME, "TEMPERATURE") == 0) {
					if (tempStart != NULL) {
						tempStart += strlen("Temperature: ");
						double temperature;
						sscanf(tempStart, "%lf%%", &temperature);
						//printf("Temperature: %.1lf°C", temperature);
						sprintf(temp_val, "Temperature: %.1lf°C", 
temperature);
						for (i = 0; i < sizeof(temp_val); i++)
						{
							if (info->encryption == Encryption_XOR)
```