Machine to Machine (M2M) Communication Using ESP8266

AMRNVB Pethiyagoda
Department of Computer Engineering
General Sir John Kotelawala Defence University
Rathmalana, Srilanka
nadinpethiyagoda4@gmail.com

OR Zain
Department of Computer Engineering
General Sir John Kotelawala Defence University
Rathmalana, Srilanka
oshinzain@gmail.com

VR Weerasekara
Department of Computer Engineering
General Sir John Kotelawala Defence University
Rathmalana, Srilanka
rusiru5707@gmail.com

***Abstract*— Machine-to-Machine (M2M) communication is a method that allows two machines to communicate with each other and exchange data without the need for human intervention. The aim of this experiment is to design and implement a low-cost M2M communication system based on two ESP8266 NodeMCU boards. One as an Access Point (Server) and another as the Station (Client). A temperature sensor is connected to client node to measure real time temperature from environment, a threshold value is configured and once exceed it should communicate with the server node which will operate the relay (actuator), this can be turned on/off. We have also conducted a distance analysis of the system to find an estimate for communicating distance between the two nodes. Further we tested the power consumption of the system and applied a power saving technique for efficiency. We estimated the maximum communicating distance as 8.2 m and were able to reduce the power consumption from 87.8 mA to 11.1 mA achieving an improvement of 87.36%.**

Keywords*— *NodeMCU, ESP8266, Received Signal Strength Indicator (RSSI), Curve Fitting

I. INTRODUCTION

M2M is crucial in Industrial Internet of Things (IIoT) systems, M2M enables nearly any sensor to communicate, opening up the idea of systems monitoring themselves and automatically responding to changes in the environment with minimal human intervention. The following experiment demonstrates the same terminology. NodeMCU is a low-cost open source IoT platform embedded with the ESP8266 Wi-Fi SoC. This Wi-Fi enabled microcontroller is capable of connecting to a Wi-Fi network, setup its own network or allow other devices to directly connect to it. Therefore, making it possible to demonstrate M2M communication between two ESP8266 NodeMCU boards. One ESP8266 as an Access Point (Server) and another ESP8266 as a Station (Client). A temperature sensor is connected to the NodeMCU configured as the Client. This system will measure the real-time temperature in the environment. The threshold value for temperature is set as per requirement, if the temperature were to exceed the threshold it communicates with the NodeMCU configured as the Server and should operate the actuator. The two devices are able to exchange data via HTTP requests. When the Server receives the notification, the actuator (relay) attached to the node should turn on. The actuator should turn off once the temperature value drops below the threshold value. Finally, this experiment also demonstrates an analysis of the range these two nodes can communicate and to apply a power saving technique to the system. The project objectives are mentioned as follows:

- To exchange data with each other via HTTP requests without connecting to the internet.
- To read real-time temperature values through a temperature sensor.
- To turn on/off the actuator when the temperature exceeds a threshold value.
- To determine a technique to reduce power consumption.
- To perform an analysis on the communicating distance between the two nodes.

The rest of this report is organized as follows. In Section II, we present our design and implementation of this experiment.

II. DESIGN AND IMPLEMENTATION

Fig 1 illustrates the system overview consisting of two machines, Machine A (Client) represents the NodeMCU ESP8266 connected with a DHT 11 temperature sensor. The function of the client node is to read temperature from the environment and determine if the value has exceeded the threshold, if so, communicate with the server. Machine B (Server) also represents the NodeMCU ESP8266, and it is connected to a 5V 5 Pin DC relay as shown. The function of the server node is to receive the notifications from the client and decide whether to turn on/off the actuator. The connection is wireless and TCP/IP (Transfer Control Protocol/Internet Protocol) is a session establishment protocol between client and server and maintained until the application data at each end have completed exchange.

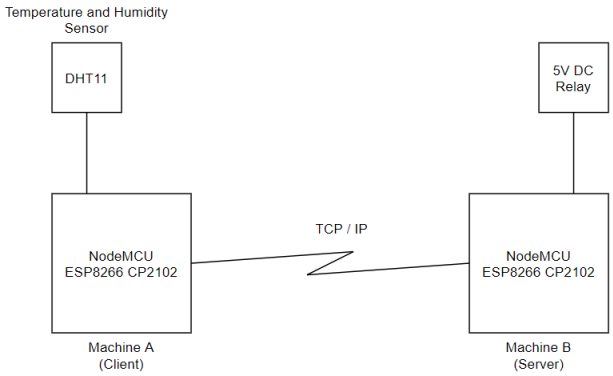

Fig 1: System overview

Following Fig 2 illustrated the circuit design of the overall system. Our primary concern is to design this system to run on external power source. Therefore, we used 9V carbon-zinc batteries to power the NodeMCU.

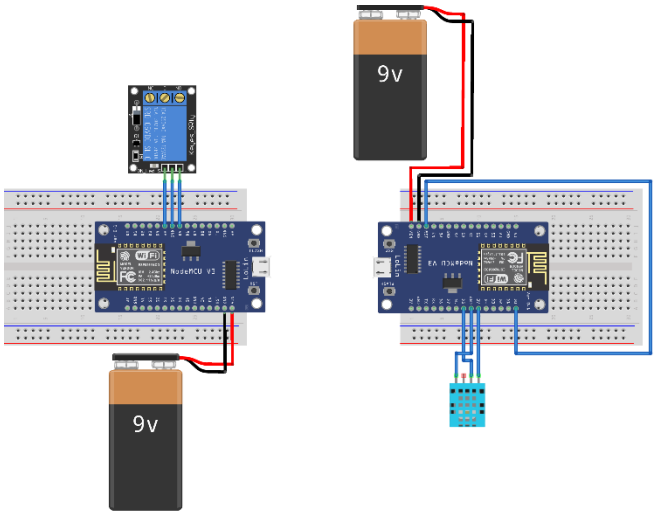

Fig 2: System circuit design

Program code 1 – Configure NodeMCU as an Access Point

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
const char *ssid = "ESPSoftAP";
const char *password = "espsoftap";
ESP8266WebServer server(80);
void handleSentVar() {
    server.send(200, "text/html", "Data received");
  }
}
void setup() {
  //Soft AP Setup
  Serial.begin(115200);
  Serial.println();
  Serial.print("Configuring Soft Access Point...");
  WiFi.mode(WIFI_AP);
  Serial.println(WiFi.softAP(ssid, password) ? "Ready" : "Failed!");
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
  server.on("/data/", HTTP_GET, handleSentVar);
  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
  server.handleClient();
}
```

Program code 2 – Configure NodeMCU as Station

```
#include <ESP8266WiFi.h>
const char *ssid = "ESPSoftAP";
const char *password = "espsoftap";
void setup() {
  Serial.begin(115200);
  delay(10);
  //Connect to the Soft-Access Point
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //Set to Station Mode
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {
}
```

Program code 3 – Configure NodeMCU as an Access Point connected with 5V DC relay

```
void handleSentVar() {
  Serial.println("handleSentVar function called...");
  if (server.hasArg("relay_state")){
    Serial.println("Change State received...");
    String relayState = server.arg("relay_state");
    Serial.print("Relay State: ");
    Serial.println(relayState);
    operateRelay(relayState); //Change actuator state
    Serial.println();
    server.send(200, "text/html", "Data received");
  }
}
void operateRelay(String state){
  if(state.equals("ON")){
    digitalWrite(14, LOW); //Turn ON actuator
  }else{
    digitalWrite(14, HIGH); //Turn OFF acutator
  }
}
void setup() {
  pinMode(14, OUTPUT); //Relay pin setup
}
void loop() {
  Serial.printf("Stations connected = %d\n", WiFi.softAPgetStationNum());
  delay(500);
  server.handleClient();
}
```

Program code 4 – Configure NodeMCU as a Station connected with DHT 11 Temperature sensor

```
#include "DHT.h"
#define DHTTYPE DHT11
#define dht_dpin 14 //D5 pin
DHT dht(dht_dpin, DHTTYPE);
void setup() {
  dht.begin();
  Serial.println("Temperature\n\n");
}
void requestServer(String state){
  WiFiClient client;
  const char * host = "192.168.4.1";
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }
  String url = "/data/";
  url += "?relay_state=";
  url += state;
  Serial.print("Requesting URL: ");
  Serial.println(url);
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }

  }
  Serial.println();
  Serial.println("Closing connection");
  Serial.println();
  Serial.println();
  Serial.println();
}
void checkTemperature(float temp){
  if(temp > 33.00){
    Serial.println("High Temperature: "+String(temp)+" C ");
    requestServer("ON");
  }
  else{
    Serial.println("Low Temperature: "+String(temp)+" C ");
    requestServer("OFF");
  }
}
void loop() {
  checkTemperature(dht.readTemperature());
  delay(500);
}
```

A. *Distance Analysis*

Analysis of the distance between the two communicating nodes were conducted and showed promising results. There are many ways to determine the distance between two nodes. We used the Received Signal Strength Indicator (RSSI) method to determine the distance. The RSSI values were obtained at specific distances and used the [1] Curve Fitting Technique (CFT) to find an equation that can estimate the distance between the two nodes. We configured the client node to measure the RSSI value from the server node and transmit the values back to the server so that it would can be displayed in the Serial window. We placed the Server node fixed at a certain point provided with constant power from the USB cable. The client node is placed at different distances growing far apart from the server node. A meter rule was used to measure the distance between the two nodes and the meanwhile the RSSI value at that distance was recorded. This process is continued until the client node disconnected with the server. The following Table I shows the recorded distances and the corresponding RSSI values.

TABLE I. DISTANCE AND RSSI VALUES

Distance (m)	***RSSI (dBm)***
0.2	-43
0.4	-49
0.6	-53
0.8	-55
1	-59
1.5	-62
2	-64
2.5	-66
3	-70
3.5	-73
4	-77
4.5	-80
5	-82
5.5	-84
6	-86
6.5	-88
7	-90
7.5	-92
8	-93
8.2	-95

The above data were inserted to an Excel sheet and saved as a .csv file. We used MATLAB software to plot the given data and find the curve fit. The csv file with the distance and RSSI data were imported and generated the script and then execute it to plot using the Curve Fitting Tool (CFT).

Program code 5 - Plot using CFT

```
%% Initialize variables.
filename = 'C:\Users\Nadin\Downloads\rssi_data.csv';
delimiter = ',';
startRow = 2;
%% Format string for each line of text:
formatSpec = '%f%f%[^\n\r]';
%% Open the text file.
fileID = fopen(filename,'r');
%% Read columns of data according to format string.
dataArray = textscan(fileID, formatSpec, 'Delimiter',
delimiter, 'HeaderLines' ,startRow-1, 'ReturnOnError',
false);
%% Close the text file.
fclose(fileID);
%% Allocate imported array to column variable names
Distance = dataArray{:, 1};
RSSI = dataArray{:, 2};
cftool(RSSI, Distance);
%% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID
dataArray ans;
```

Program code 6 – Final Curve Fit graph

```
function [fitresult, gof] = createFit(RSSI, Distance)
%% Fit: 'Curve Fit'.
[xData, yData] = prepareCurveData( RSSI, Distance );
% Set up fittype and options.
ft = fittype( 'exp1' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.StartPoint = [0.0336520798880578 -
0.0600756600776581];
% Fit model to data.
[fitresult, gof] = fit( xData, yData, ft, opts );
% Plot fit with data.
figure( 'Name', 'Curve Fit' );
h = plot( fitresult, xData, yData );
legend( h, 'Distance vs. RSSI', 'Curve Fit', 'Location',
'NorthEast' );
% Label axes
xlabel RSSI
ylabel Distance
grid on
```

B. *Power Consumption*

Reducing the power consumption is an essential part if we were to build these types of systems that need to operate in real world using an external power source. For testing the power consumption and to apply a power saving technique we used 9V carbon-zinc batteries as the power source. The ESP8266 module of the NodeMCU can operate in four power modes.

1) *Active mode*
2) *Modem-sleep mode*
3) *Light-sleep mode*
4) *Deep-sleep mode*

Out of the above four modes we can use the three sleep modes to reduce power consumption according to our requirements. The following table shows the power consumption when configure to the three sleep modes.

TABLE II. [2] SLEEP MODES

Item	*Modem-sleep*	*Light-sleep*	*Deep-sleep*
Wi-Fi	OFF	OFF	OFF
System clock	ON	OFF	OFF
RTC	ON	ON	ON
CPU	ON	Pending	OFF
Substrate Current	15 mA	0.4 mA	~ 20 µA
Average current (DTIM = 1)	16.2 mA	1.8 mA	–
Average current (DTIM = 3)	15.4 mA	0.9 mA	–
Average current (DTIM = 10)	15.2 mA	0.55 mA	–

Configuring the ESP8266 to Deep-sleep mode is the most power efficient option. The above table shows that the ESP8266 consumes approximately 20 µA. In this mode Wi-Fi, System clock, CPU will be turned off and only the Real Time Clock (RTC) will be operating.

Current consumption measure was done using a multimeter, As shown in Fig 3, First, we configured the ESP8266 to be working in active mode connected to a 9V carbon-zinc battery with the DHT 11 temperature sensor and multimeter. The multimeter was set to DCA at 200m to measure the current draw.

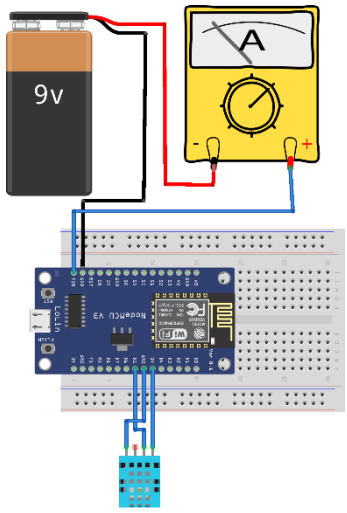

Fig 3: Circuit design to measure power consumption from client.

Next, we configured circuit in Fig 3 to Deep-sleep mode. After the ESP8266 is in Deep-sleep mode, there are [2] two ways of waking it up

1)Timer wake up – wakes up after a predefined period of time.

2)External wake up -wakes up when you press the RST button.

The former method is the most suitable, this will avoid the need for pressing the button every time the ESP8266 needs to wake up. To setup the timer wake up, the RST pin has to be connected with the D0 (GPIO 16) pin as illustrated in Fig 4. [2] The D0 pin has a wake feature and RST pin of the ESP8266 operates at HIGH voltage and when it receives a LOW signal, it attempts to restart the NodeMCU.

Program code 7- ESP8266 Deep sleep

```
ESP.deepSleep(15e6);
```

This statement will put the ESP8266 to Deep-sleep for 15 seconds.

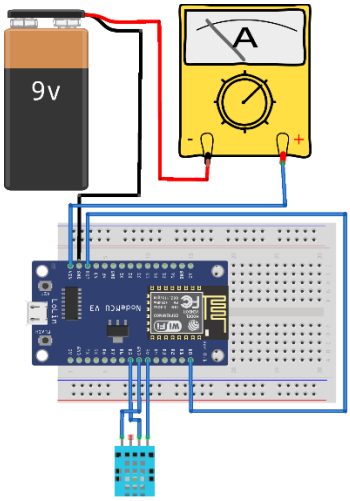

Fig 4: Circuit design to measure power consumption from client in Deep-sleep mode.

Fig 5 shows the circuit connection to measure the current from the server node connected to a 9V battery with the 5V DC relay.

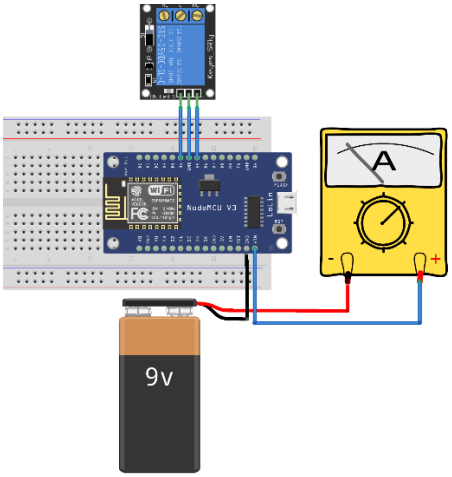

Fig 5: Circuit design to measure power consumption from server.

III. RESULTS AND DISCUSSION

The graph shown in Fig 6 produces an exponential model, the Distance vs RSSI plots can be identified and the curve represents the curve fit obtained using the model. The equation of the exponential model is generated. The equation is shown as follows:

$$f(x) = a^{\exp(b^x)}$$

The coefficients (with 95% confidence bounds) were:

$$a = 0.09372\ (0.05446, 0.133)$$
$$b = -0.04784\ (-0.0526, -0.04307)$$

Goodness of fit:

SSE: 2.798
R − square: 0.9801
Adjusted R − square: 0.979
RMSE: 0.3942

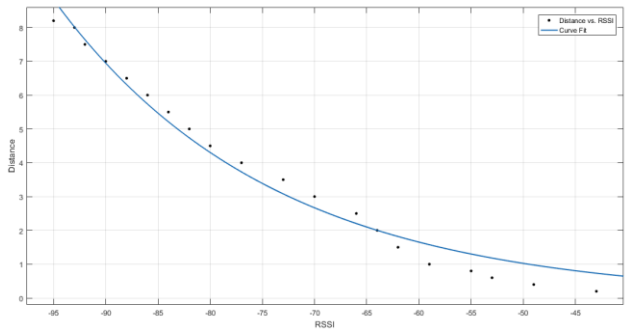

Fig 6: Curve Fitting Plot for Distance vs RSSI

For the power consumption as shown in Fig 3 the multimeter reading had maximum current consumption of 87.8 mA and a minimum current consumption of 79.1 mA. The reason multimeter readings were fluctuated because the current draw values are different for the NodeMCU to turn off the Wi-Fi, read the DHT sensor to compare values and to request the server. To calculate the runtime of the client node, the equation is used as shown below:

$$Runtime\ (hours) = \frac{Battery\ Capacity\ (mAh)}{Current\ Consumption\ (mA)}$$

The power source was a 9V carbon-zinc battery, this typically has a capacity of 400 mAh. Calculating the runtime of the system with maximum current consumption was obtained as 4.55 hours this is in the worst-case scenario. For minimum current consumption was 5.05 hours was the result which the best-case scenario.

Next, we measured the readings when configured in Deep-sleep mode as shown in Fig 4, and the current consumption was recorded as 11.1 mA. The runtime of the system was calculated as 36.0 hours.

We also measured the consumption in the server node, as shown in Fig 5 the reading was recorded as 127.5 mA, this reading was obtained when the 5V DC relay module was in "ON" state. Also, we measured the reading during the "OFF" state as well and was recorded as 80.8 mA. The runtime in the worst-case scenario was calculated to be 3.13 hours and best-case to be 4.95 hours respectively.

An alternative method of reducing the power consumption would be to change the Transmit (TX) and Receive (RX) of data exchanged wirelessly.

TABLE III. [3] RF POWER CONSUMPTION

Parameters	***Min***	***Typical***	***Max***	***Unit***
TX802.11b, CCK 11 Mbps, POUT = +17 dBm	–	170	–	mA
TX802.11g, OFDM 54Mbps, POUT = +15 dBm	–	140	–	mA
TX802.11n, MCS7, POUT = +13 dBm	–	120	–	mA
Rx802.11b, 1024 bytes packet length, –80 dBm	–	50	–	mA
Rx802.11g, 1024 bytes packet length, –70 dBm	–	56	–	mA
Rx802.11n, 1024 bytes packet length, –65 dBm	–	56	–	mA

Configuring the ESP8266 Transmit to TX802.11n and the Receive to Rx802.11b will have an impact on the reducing the power consumption of the NodeMCU. However, this may cause the communicate distance to vary and the data collected will be different during the distance analysis process.

IV. CONCLUSION

The purpose of this experiment was to design and implement a low-cost M2M communication system based on two ESP8266 NodeMCU boards. One node to act as a Station (Client) connected with a DHT 11 temperature sensor to measure real time temperature from the environment, the other node as an Access Point (Server). A threshold value is set and whenever the temperature exceeds the threshold the client should communicate with the server. Depending on the situation the server will turn on/off the actuator (5V DC relay). During distance analysis the system demonstrated a maximum of 8.2m communicating distance, this is the distance at which the Station disconnects from the Access Point. The applied power saving technique brought significant improvement to the reduce the current consumption and was able to reduce it from 87.8 mA to 11.1 mA. The system is able to achieve an improvement of 87.36 %.

ACKNOWLEDGMENT

We would like to extend our gratitude to Mr. MWP Maduranga for his kind cooperation and encouragement in completing this project.

V. REFERENCES

[1] B, Suvankar & B, Debajyoti & S, Buddhadeb, "Estimate distance measurement using NodeMCU ESP8266 based on RSSI technique", IEEE CAMA, Tsukuba, Japan. 2017, pp. 170-173.

[2] Ranjith et al, "ESP8266 Deep Sleep with Arduino IDE (NodeMCU) | Random Nerd Tutorials", Randomnerdtutorials.com, 2021. [Online]. Available: https://randomnerdtutorials.com/esp8266-deep-sleep-with-arduino-ide/. [Accessed: 18- Oct- 2021].

[3] ESP8266EX Datasheet, Espressif Systems, 2020. [Online]. Available: https://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf [Accessed 18-Oct-2021].