Report on Distributed Communication

The **unsuccessful** attempt with Arduino C/C++:

First set up your ESP32 and Arduino IDE.

Then run CODE A on your ESP32

1. CODE A is a C++ code that connects the ESP32 with the wifi network and then prompts the user to input a message and then the message is sent to a server.

| // CODE A  #include <WiFi.h>  const char\* ssid = "SWAN LAB";  const char\* password = "swan@123";  const char\* host1 = "192.168.1.24"; // IP address of Server 1  const int port1 = 5005; // Port of Server 1  const int localPort = 4095; // Local port to listen for incoming connections  WiFiClient client; // Client for Server 1  WiFiServer server(localPort); // Server to listen for any incoming connection  String lastSentMessage = ""; // Store the last sent message  unsigned long myTimerStart = 0; // Store the start time of the timer  void setup() {  Serial.begin(115200);  WiFi.begin(ssid, password);  while (WiFi.status() != WL\_CONNECTED) {  delay(500);  Serial.print(".");  }  Serial.println("WiFi connected");  if (!client.connect(host1, port1)) {  Serial.println("Connection to Server 1 failed");  return;  }  Serial.println("Connected to Server 1");  server.begin(); // Start the server to listen for any incoming connections  Serial.println("Server started");  }  void loop() {  // Send data to Server 1  if (client.connected()) {  // Check if there's any input from the Serial Monitor  if (Serial.available() > 0) {  // Read the input as a string  String message = Serial.readString();    // Send the input as a message to the server  client.print(message);  // Store the sent message and start the timer  lastSentMessage = message;  myTimerStart = millis();  }  } else {  Serial.println("Re-connecting to Server 1");  if (!client.connect(host1, port1)) {  Serial.println("Connection to Server 1 failed");  return;  }  Serial.println("Connected to Server 1");  }  // Check if there are any new clients  if (WiFiClient newClient = server.available()) {  // If a new client connects, print a message and read their data  Serial.println("New client connected");  while (newClient.connected() && newClient.available()) {  String line = newClient.readStringUntil('\r');  Serial.print("Received: ");  Serial.println(line);  // If the received message is the same as the last sent message, stop the timer and print the elapsed time  if (line == lastSentMessage) {  unsigned long elapsedTime = millis() - myTimerStart;  Serial.print("Elapsed time: ");  Serial.print(elapsedTime);  Serial.println(" ms");  }  }  }  } |
| --- |

The above code was successfully connecting to the swan lab wifi and connecting to a python server and forwarding the received message to it.

2)

//CODE B- this code is a python server which uses socket programming of python and acts as a server and receives any message it receives from a client that sends it any message over the local network and also forwards it to the next server.

**import socket**

**# Create a TCP/IP socket**

**sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**# Define the server host and port**

**host = '192.168.1.24' # The IP address of your laptop**

**port = 5005 # The same port as used by the ESP32**

**# Bind the socket to the host and port**

**sock.bind((host, port))**

**# Listen for incoming connections**

**sock.listen(1)**

**print(f'Server started on {host}:{port}')**

**# Define the forwarding server host and port**

**forward\_host = '127.0.0.1' # The IP address of the next server**

**forward\_port = 6000 # The port of the next server**

**while True:**

**# Wait for a connection**

**print('Waiting for a connection...')**

**connection, client\_address = sock.accept()**

**try:**

**print(f'Connection from {client\_address}')**

**# Receive the data in small chunks and print it**

**while True:**

**data = connection.recv(16)**

**print(f'Received "{data.decode()}"')**

**if data:**

**print('Sending data back to the client')**

**connection.sendall(data)**

**# Forward the data to the next server**

**print('Forwarding data to the next server')**

**forward\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**forward\_sock.connect((forward\_host, forward\_port))**

**forward\_sock.sendall(data)**

**forward\_sock.close()**

**else:**

**print('No more data from', client\_address)**

**break**

**finally:**

**# Clean up the connection**

**connection.close()**

The above code was capable of receiving the message from the esp32 running on an arduino ide and printing it on the screen. The above code was also successful in forwarding the received message to a next server .

3) //CODE C :

import socket

import threading

import time

class Node:

def \_\_init\_\_(self, host, port):

self.host = host

self.port = port

def start\_server(self, forward\_host, forward\_port):

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((self.host, self.port))

server.listen(1)

print(f"Server started on {self.host}:{self.port}")

while True:

client, address = server.accept()

data = client.recv(1024)

if not data:

break

print(f"Received: {data.decode()}")

client.send("Message received".encode())

time.sleep(5)

self.forward\_message(data.decode(), forward\_host, forward\_port)

def forward\_message(self, message, host, port):

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect((host, port))

client.send((message + '\r').encode())

if \_\_name\_\_ == "\_\_main\_\_":

node4 = Node('localhost', 7050)

threading.Thread(target=node4.start\_server, args=('localhost', 4095)).start()

This code was able to receive any message from the server and print it on screen but failed every time with any sort of modification to forward the received message to an esp. Please note that whenever you try to send a message to an esp32 running on arduino ide, always append the message with (‘\r’) because the arduino ide keeps reading for incoming messages until it encounters an (‘\r’) in the incoming string.

No other code on an esp32 was successful in receiving any message from the above server(CODE C).

This was the **successful** attempt using Micropython:



We have used Thonny IDE for writing and executing the codes that will run on the ESP32 and normal notebook to write and execute the python codes that act as servers.

Please note that the black usb wires that are provided with the esp32 are highly inefficient and lead to errors like “burnout “. We had to replace the given usb cables with our own usb cables to complete the work.

Once we write our code on the thonny IDE we click on "run" on top of the screen and select “manage interpreters' ' and then we select our desired port and language. If you have not installed micropython yet you get an option to download it typically at the bottom right corner of your screen.

After that you need to click on the green play button and the code will be pushed to your esp32 and your esp32 should be able to connect to the wifi network. As the “shell” prompts the user to input a message , you write something and watch it come back through several servers concatenated with the ip address of the esp32. Below is an explanation on how exactly the code works and important changes that you need to do when replicating the result.

1. **Running** send\_hello\_client\_to\_server1.py **in Thonny**:

The send\_hello\_client\_to\_server1.py script is designed to run on an **ESP32** device.It connects to a WiFi network named “SWAN LAB” with the password “swan@123”.Once connected, it sends an HTTP POST request to the server at the address “http://192.168.1.24:5002/send\_message”.The payload of this request includes a message ID (presumably 1) and a formatted message.If a valid response is received (not equal to “No message for the specified ID”), it prints the received message and forwards it to the second server at “http://192.168.1.24:5000/esp32”.The loop repeats every 5 seconds.

After running the first esp32 you open the command prompt and execute the below code, as you can see the command prompt will print something in this format $192.164.1.24$hello world$ 200. Let us explain what all this is . The $ is a delimiter. “ hello world” is the message you put it. The number “192.164.1.24” is the ip address of the esp32 from which the message is being sent from. And lastly the number 200 is a random number that is generated every time a new message is sent out from esp32. The random numbers generated range from 0 to 255. Below is an explanation on how the code works internally and any changes that you might need to do when replicating the result.

1. **Running servers1.py in CMD**:

The severs1.py is a Flask application running on a Windows machine.It listens for incoming HTTP POST requests at the endpoint “/send\_message”.When it receives a request, it extracts the message ID and checks if a corresponding message exists in its internal dictionary.If a message is found, it modifies the message (appending "Send from server 2: ") and responds with the modified message. If no message is found, it responds with “No message for the specified ID”.

Please note that the IP address that is shown on the command prompt when running this server is the IP address of the ESP32 that lies 2nd in the relay. Let me explain, there are 2 ESPs, the first one is the one which receives messages and forwards. There is another ESP right ? The second ESP32, yes, that one. The second ESP sends a POST request to servers1.py and every time the servers1.py receives a request from ESP32 , it prints the IP address of the ESP32. And the ESP32 just prints whatever is printed on the servers1.py. The servers1.py here acts as a postbox, the first esp32 sends some message to this server and the 2nd esp looks continuously for any new message in the servers1.py and once it receives a message , the 2nd esp just copies it and prints the message in its own shell .

1. **Running received\_from\_server1\_send\_to\_server2.py in Thonny**: Similar to send\_hello\_client\_to\_server\_1.py , the received\_from\_server1\_send\_to\_server2.py script runs on an ESP32 device.It connects to the same WiFi network (“SWAN LAB”).It prompts the user to input a message.The ESP32’s IP address, a random number, and the user’s input are combined into a formatted message.This message is sent to the first server (“http://192.168.1.24:5002/send\_message”).The script then enters a loop to continuously check for messages from the second server (“http://192.168.1.24:5000/esp32”).If a valid message is received, it prints the message and breaks out of the loop.

This esp32 theoretically knocks at servers1.py continuously and once it has something new , the esp32 grabs it and prints it. Not only that, it also forwards the printed message to the next server that is servers1.py. Please note that you can just alter the ip address and port number and copy paste the code of the 2nd esp32 and make as many intermediate nodes as possible and connect as many esp you want.

4) **Running servers2.py in CMD**:

The servers2.py script is another Flask web application running on the same Windows machine.It listens for incoming HTTP POST requests at the endpoint “/esp32”.When it receives a request, it extracts the message ID and the actual message.It stores the message in an internal dictionary.The script also listens for requests at the endpoint “/send\_message”, similar to the first server.If a valid message ID is provided, it retrieves the corresponding message, modifies it, and responds accordingly.

The ESP32 devices (clients) connect to the WiFi network, send messages to the first server, which then modifies and forwards the messages to the second server. The second server responds to requests from both clients.