|  |
| --- |
| **Emulating a TCP-Over-UDP Connection Using an ESP32 Dev Board**  CS 5283-50 Computer Networks  Team Members: Brian Goldsmith & Damon Raynor |

1. Purpose

The purpose of this project is to explore Wi-Fi data communication over the internet from one physical device (our personal CPU/laptop) to another (an ESP32 Development Board). Our main goals are to:

1. Visually demonstrate the TCP-over-UDP Handshake protocol.
2. Successfully send messages from one physical device to another over Wi-Fi.
3. Observe and capture the traceroute of relevant network traffic between the physical devices.

In order to meet our goals, we went through the process of procuring, programing and setting up an ESP32 Development board and ancillary electrical components (to aid in visualizing successful connections and transfers of data). The remainder of this report details (1) our approach to choosing and configuring our hardware setup, (2) programming the client and server comms channel, and (3) analyzing the results of our traceroute captures.

1. Project Description
   1. High level Project Architecture

This project consisted of communicating to an Arduino-like ESP32 Development board from a personal computer over Wi-Fi. The decision was made to assign the ESP32 board as the server that the personal computer would request a connection from. Table 1 identifies the network protocol suite that governs the project.

Table 1: Network Protocol Suite

|  |  |
| --- | --- |
| **Network Layer** | **Protocol** |
| Application | Custom (TCP-Like) |
| Transport | UDP |
| Network | IP |
| Link | Wi-Fi |

The custom application layer protocol was designed to ensure reliable data transfer over UDP via (1) implementing the TCP handshake protocol (reference Figure 1 for a visualization) (2) acknowledging receipt of packets sent and (3) implementing the TCP protocol for closing the connection after all information has been sent (reference Figure 2).

Diagram

Description automatically generated

Figure 1: TCP Three-Way Handshake visualization

Diagram

Description automatically generated

Figure 2: TCP Closing Connection Visualization

* 1. Verification Strategy

To verify that the application worked as intended, (1) the ESP32 Development board was outfitted with two LED light bulbs and programmed to turn the LEDs on and off to visualize the establishment and closing of a connection, (2) the client and server side of the application was programmed to print their respective status to the terminal, and (3) the network monitoring software, Wireshark, was used to trace the communication between the two devices. Figure 3 defines the expected behavior of each LED to visualize each step of the communication process.

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Server Connection State** | **LED 1 State** | **LED 2 State** |
| **0** | No Connection | Off | Off |
| **1** | LISTEN | Blinking | Off |
| **2** | SYN RCVD  (Handshake Initiated) | Steady | Off |
| **3** | ESTAB  (Connection Established) | Steady | Steady |
| **4** | Receiving Message | Steady | Off |
| **5** | CLOSE WAIT | Off | Blinking |
| **6** | CLOSED | Off | Off |

Figure 3: LED State Representation of the Communication Process

Refer to section 3 for a detailed description of the hardware setup, section 4 for the application code architecture and section 5 for the conclusion, code outputs received and Wireshark traces captured.

1. Embedded Device Description and Setup
   1. ESP32 Development Board High Level Overview

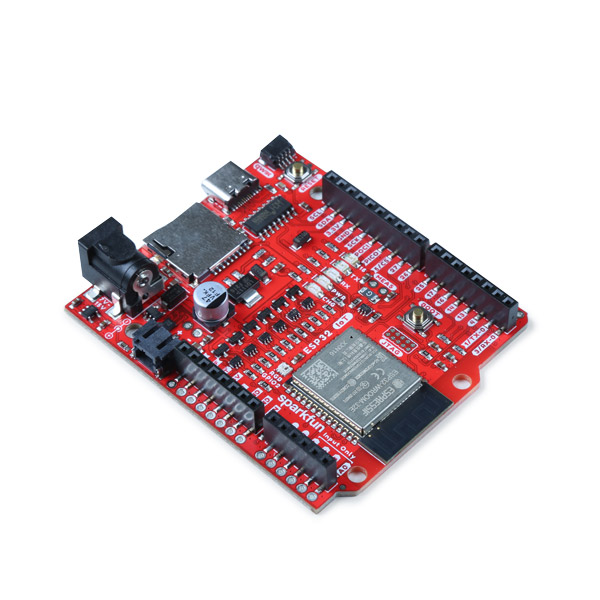


Figure 4: ESP32 Development Board

According to the manufacturer, ‘the SparkFun IoT RedBoard is an ESP32 Development Board that includes Espressif's ESP32 WROOM, Wi-Fi and Bluetooth® MCU module that targets a wide variety of applications. At the core of this module is the ESP32-D0WDQ6 chip which is designed to be both scalable and adaptive. The IoT RedBoard can target a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming, and MP3 decoding.’

This board was chosen primarily for its Wi-Fi capabilities and affordable price point. The following link routes to the website where you can purchase this board and read more about its specifications and potential use cases: [SparkFun IoT RedBoard - ESP32 Development Board - WRL-19177 - SparkFun Electronics](https://www.sparkfun.com/products/19177)

This ESP32 board is powered and programmed via a USB to USB-C cable. The Arduino IDE was used for writing and uploading code to the ESP32. The Arduino Programming Language was necessary to properly program the board – which is basically a framework built on top of C++.

Once powered, this board and other Arduino-like boards run continuously on a loop of whatever has been uploaded to it. The programming structure to support this generally consists of two sections/functions: (1) **setup()** and (2) **loop()**. When powered, the board will initially run **setup()** only once, and then will continuously loop though the **loop()** function. It is necessary to plan how you will execute your program within this programming structure. Section 4 details the application code architecture for this project.

* 1. Hardware Setup and Circuit Layout

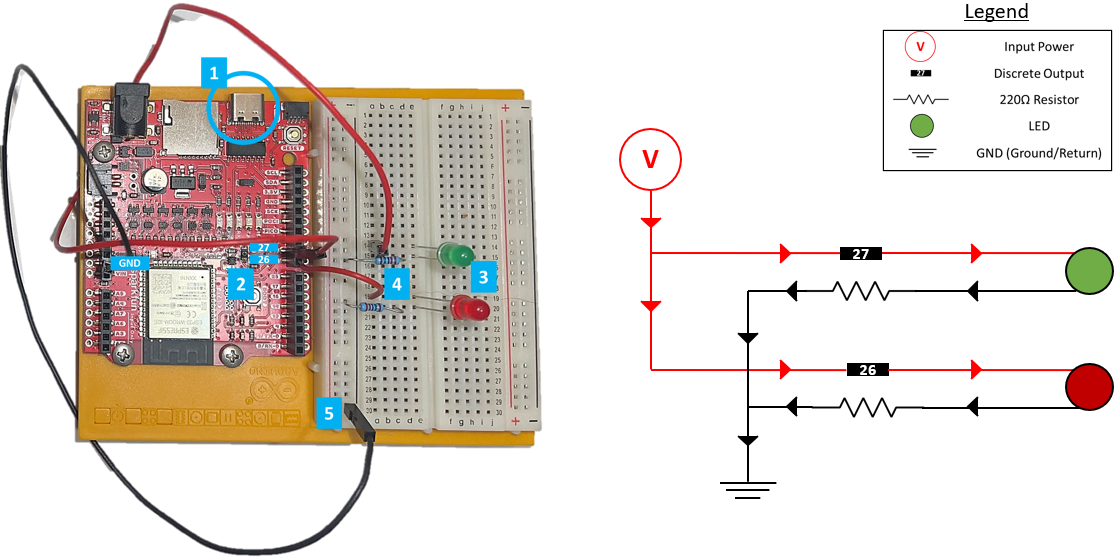


Figure 5: Hardware Setup and Circuit Layout

Figure 5 shows the hardware setup and circuit layout used for this project. Our hardware consisted of the following components:

* (1) ESP32 Development Board
* (1) Circuit Bread Board
* (2) LEDs
* (2) 220Ω Resistors
* (3) Wires

Referencing the blue numerical indicators depicted in Figure 5, power comes through the USB-C port (1), outputs 3.3V at discrete outputs ‘26’ and ‘27’ (2), which is directed to the LEDs – each discrete output controls one LED – (3), then is reduced by a 220Ω resistor (4), and finally grounded to complete the circuit (5).

There are a total of two circuits. Each circuit shares the same power source, has one discrete output, one LED, one resistor and shares a common ground. Figure 5 also shows a simplified schematic of each circuit. The red lines/arrows represent the power input path while the black lines/arrows represent the return path.

This setup was inspired by the following tutorial: [ESP32 Web Server - Arduino IDE | Random Nerd Tutorials](https://randomnerdtutorials.com/esp32-web-server-arduino-ide/)

1. Application Code Architecture

The application code for this project was heavily influenced by Programming Assignment 2 from the Vanderbilt University, CS 5283-50 Computer Networks, Fall 2022, course. The client code was only modified to identify the address of the ESP32 board and remained housed on our personal computer. The server code was ported over to the ESP32 board and heavily modified to accommodate for the expected programming structure and language as mentioned in section 3.1.

The following considerations and steps were taken to successfully refactor the server-side code:

TODO:

* Program language considerations
* Setup and loop functions
* Adding delays
* Configuring LEDs on, off, steady, blinking
* Etc.

1. Project Results
   1. Conclusion

By way of our verification strategy, we were able to successfully meet our goals as outlined in section 1.

1. We were able to visually demonstrate our intended communication process via controlling the two LEDs as described in section 2.2.
2. The terminals for both the client and server-side displayed the expected output which identified that a connection was established, a message was broken up, sent in packets, and correctly reassembled. See section 5.2 for the client and server printouts.
3. Wireshark properly detected and reported on the activity between both devices. Section 5.3 contains the Wireshark log file that shows communication between both devices.
   1. Expected Client and Server Terminal Outputs

Attached to this report are the client and server logs outputted after establishing a connection, sending the message *“It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness”*, and closing the connection.

|  |  |
| --- | --- |
| **Client Logs** | **Server Logs** |
|  |  |

Using Figure 6 as an example, we can confirm that (1) a Wi-Fi connection was established, (2) that the handshake protocol was a success, and (3) that packets containing the message has begun to be received. Reviewing the Client Logs will reveal that the server properly acknowledges that packets have been received and that the connection closes properly.

Text

Description automatically generated

Figure 6: Server Logs Snippet

* 1. Observed Network Traffic

Attached is the Wireshark capture log associated with the communication from our personal computer to the ESP32 Board. The IP address for the computer is 192.168.1.11 and the IP address for the ESP32 Board is 192.168.1.33. Figure 7 shows the results when filtering on “ip.addr == 192.168.1.33”. This confirms communication between both devices.

|  |
| --- |
| **Wireshark Logs** |
|  |

Table

Description automatically generated

Figure 7: Wireshark Logs Filtering on “ip.addr == 192.168.1.33”