# Practical Sheet Nº 3 ( Sockets TCP)

| Objectives | Duration |
|---|---|
| By the end of the training, you will be able to:<br>• Become familiar with client-server communications using TCP sockets;<br>• Implement a TCP socket server in python;<br>• Implement a TCP socket client using ESP32s DEVKIT<br>• Implement an Ad Hoc protocol for exchanging messages between devices.<br>• Understand the main differences between TCP and UDP sockets. | 30min. |

## Part 1 – Introduction and preparation

In this practical sheet we intend to develop a client/server application with the objective of exchanging messages between devices at the transport layer level (Fig.1). A UDP (User Datagram Protocol) or TCP (Transmission Control Protocol) communication normally uses two applications (client and server) configured to exchange messages encoded in binary or text (ASCII). In this practical work the client and server applications will be developed in Arduino C (ESP32s DEVKIT) and in Python, respectively. Python is an interpreted language of very high-level language (VHLL), cross-platform (Linux, Windows, and Mac OS), object-oriented oriented and open source.

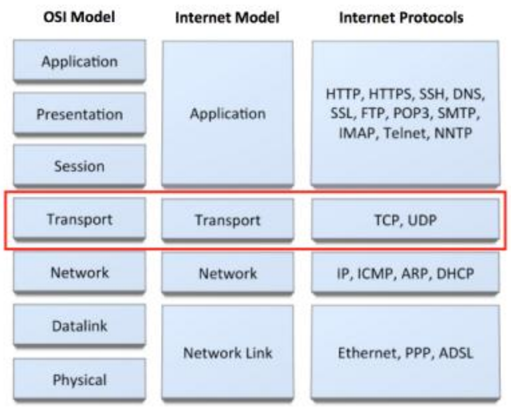


*Figure 1: OSI Model*

For those who have never programmed in Python, the following links are recommended:

- **Installation:** https://wiki.python.org/moin/BeginnersGuide/Download
- **Beginners Guide**: https://www.python.org/about/gettingstarted/
- **Tutorial:** https://wiki.python.org/moin/BeginnersGuide/Programmers

To improve performance in practical classes, we recommend that you read and run the following tutorial to familiarize yourself with the syntax used in Python:

http://www.tutorialspoint.com/python/index.htm

The goal of the class is to establish a communication between client and server by defining an ASCII encoded Ad-Hoc protocol that allows bi-directional messages to be sent between client and server, as defined in the following image:

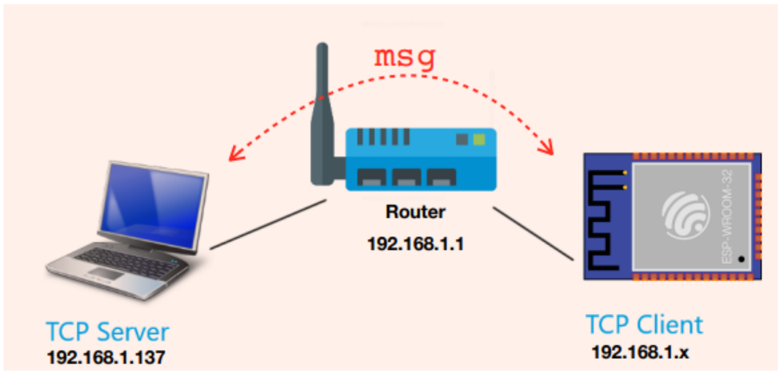
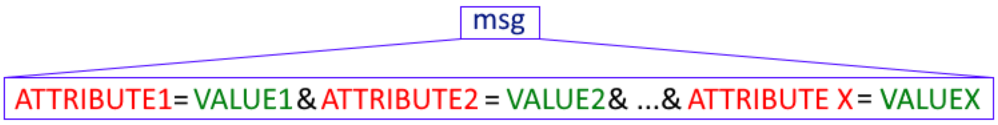


*Figure 2; Ad-hoc protocol to exchange messages between server and client.*

The frame structure to be used in the serial byte-to-byte communication between the two devices uses text (see ASCII table here) and has the following format:


msg
ATTRIBUTE1=VALUE1&ATTRIBUTE2 = VALUE2& ...& ATTRIBUTE X= VALUEX

In the frame entered before, the following characters are used to identify:

- **'&'** - ATTRIBUTE/VALUE pair separator
- **'='** - Divider of the ATTRIBUTE/VALUE pair

To distinguish the different devices, we will use an attribute to identify which device (ID) In this approach, multiple attribute/value pairs can be added, for example.

- **STATE**: to identify a state
- **VAR:** to identify the value of a quantity or variable

The ID attribute values identify which device, for example '0', '1', '2', etc., and the following attribute-value pairs may represent the new state (STATE) in which to place a given device***: '0' - OFF*** and ***'1' - ON***.

For example, the following is a frame (in ASCII) that can be used to set the device to ***ID=0*** with ***STATE=OFF:***

ID=10&STATE=0

Another example, below is a frame (in ASCII) that can be used to send a value a quantity or variable ***ID=7*** with the ***VAR=34***

ID=10&VAR=34

## Part 2 – Lab Component

In this part you want to implement a socket server in Python that will run locally on your computer. Afterwards you want to develop a client application in Arduino C to run on the ESP32s DEVKIT. The diagram in Figure 3 shows the complete sequence of methods called by the client and server applications in the communication process.

For detailed information about the methods used in the socket class in Python refer to this link.

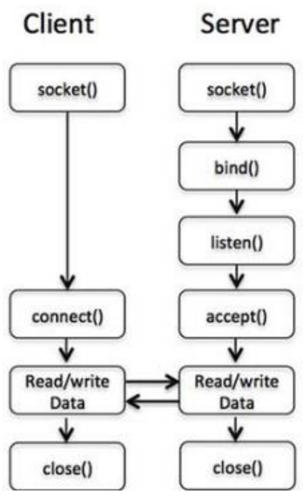


*Figure 3: TCP Sockets*

### Laboratory procedure

1. Consider the following code to implement the client in ESP32s DEVKIT. Adapt the code so that the client connects to the server implemented earlier on port 8090. Using the text editor of the Arduino IDE create a sketch named **FP2_ESP32_client.ino.**

```
#include <WiFi.h>
#define ID_KIT 10 // Group IDentifier
const char* ssid = “cmm"; // Network SSID
const char* password = “0123456789"; // Network Password
// To get the server IP use this commands
// Windows: use ipconfig command
// Linux/Mac: use ifconfig command
const char* host = “___.___.___.___”; // Server IP Address
const int port = 8090 + ID_KIT; // Server Port
// Setup Function - Inits Serial Port and WiFi Connection
```

```
void setup() {
      Serial.begin(115200); // Init Serial Port
      Serial.print("Connecting to ");
      Serial.println(ssid);
      WiFi.begin(ssid, password); // Connects to WiFi Network
      while (WiFi.status() != WL_CONNECTED){ // Waits for WiFi connection
            delay(500);
            Serial.print(".");
      }
      Serial.println("");
      Serial.println("ESP connected to WiFi with IP address: ");
      Serial.println(WiFi.localIP());
}
```

2. Prepare the message to be sent to the server according to the Simple Ad Hoc Protocol defined in the introduction:

   ***ID=10&VAR=-13***

   The value to be sent should be a randomly generated integer belonging to the range [-10 10]. The message msg is then arranged and encoded in ASCII format using the existing String class in Arduino C, as exemplified below:

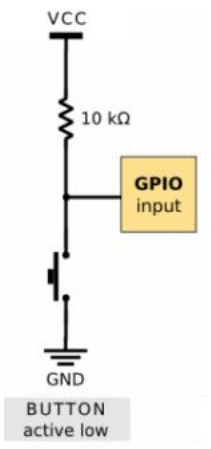
```
int value = random(-10,10);
String msg = "ID=" + String(ID_KIT) + "&VAR=" + String(value);
```

3. Given the message definition adapt the following code to implement the client loop function, which will run on the ESP32s DEVKIT.

```
// Loop Function
void loop() {
      delay(5000); // Waits 5 seconds
      Serial.print("Connecting to "); // Forces a new connection
      Serial.println(host);
      WiFiClient client; // Creates a TCP connection
      // with WiFiClient class
      if (!client.connect(host, port)) {
            Serial.println("Connection failed!”);
            return;
        }

        // Ready to send Data to the server
        int value = _____________; // Generates a random value
        String msg = ____________________________ // Prepares the msg to send
        Serial.print(" > Sending Message: ");
        Serial.println(msg); // Prints msg in console
        client.print(msg); // Sends msg to server
        client.stop(); // Closes TCP Socket
    }
```

4. Test the client code you have implemented by connecting to the socket server available for this purpose. You should connect to port ***8090 + ID_KIT***. Request the server's IP.
   Now add a push button as illustrated in the assembly shown in Figure 4.



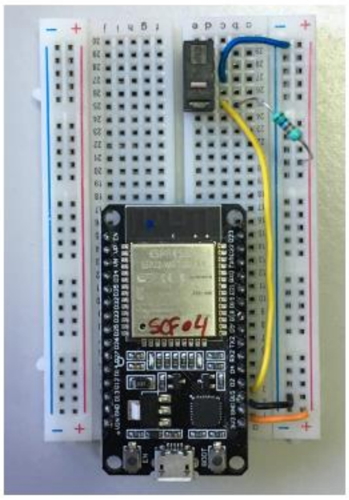


***Figure 4: Pull up Button circuit***

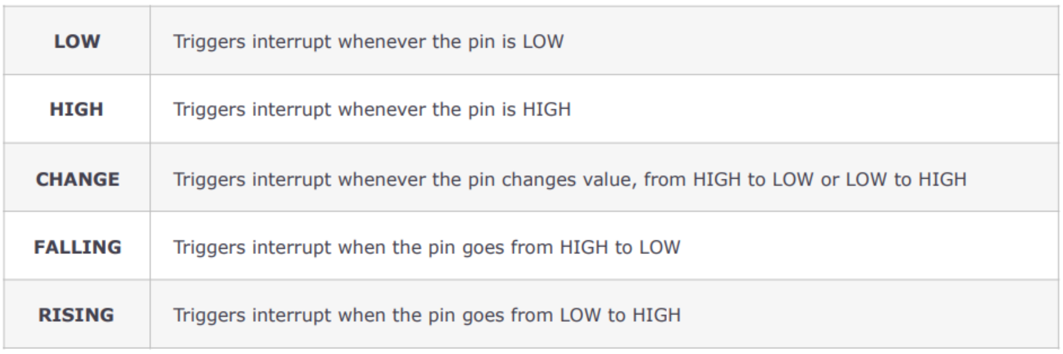
5. Make a copy of the program implemented in three and call the new program ***FP2_EX5.ino***. Test the button by adapting the new program so that it transmits the following message, considering the button's state:
   - **Button Clicked**: ID=10&VAR=10
   - **Button Not Clicked**: ID=10&VAR=-10

6. Make a copy of the program implemented in 3) and call the new program FP2_EX6.ino. Design a new program that proceeds according to the following application requirements:
   - Count the number of clicks the user makes per second;
   - The counting should be done using external interrupts using an interrupt service routine (myISR).
   - The message transmission period should be one second;
   - The message format shall be as follows: ID=10&VAR=NumClicks

   To configure the external interrupts for a specific GPIO on the ESP32 in the setup function configure the interrupt as follows:

```
attachInterrupt(digitalPinToInterrupt(GPIOpin), my_BTN_ISR, Mode)
```

where GPIO represents the pin that is intended to be used to generate the interrupt; myISR represents the interrupt service routine; and Mode the way the interrupt will operate, as you can observe in the following table.



| | |
|---|---|
| **LOW** | Triggers interrupt whenever the pin is LOW |
| **HIGH** | Triggers interrupt whenever the pin is HIGH |
| **CHANGE** | Triggers interrupt whenever the pin changes value, from HIGH to LOW or LOW to HIGH |
| **FALLING** | Triggers interrupt when the pin goes from HIGH to LOW |
| **RISING** | Triggers interrupt when the pin goes from LOW to HIGH |

Finally, add a Service Interruption Routine (SIR) that ensures that no multiple occurrences are not counted in a single click.

```
// Interrupt Service Routine - my_BTN_ISR
void my_BTN_ISR() {
static unsigned long last_interrupt_time = 0;
unsigned long interrupt_time = millis();
// interrupts faster than 50 ms, assume it is a bounce and ignore
if (interrupt_time - last_interrupt_time > 50){
// PUT YOUR CODE HERE
}
```

```
        last_interrupt_time = interrupt_time;
        }
    }
```

7. Now implement your own server application. Consider the following code to implement a socket server in Phyton that should run on port 8090 + ID_KIT of your computer. Using a text editor create a script named ***TCPserver.py.***

```
import socket
ID_KIT = ___
server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host = "localhost"
port = 8090 + ID_KIT
server.bind((host,port))
server.listen(1)
while True:
clientsocket, addr = server.accept()
print "----------------------------------------------------"
print "Client has address",clientsocket.getsockname()
print clientsocket.recv(100)
print "Received from",addr
print "----------------------------------------------------"
 clientsocket.send(“MSG:RX:OK”)
clientsocket.close()
```

8. Using the terminal execute the script by typing:

```
$ python TCPServer.py
```

9. Given a real application, identify up to 2 more Attribute/Value pairs. Change the format of the message and test the communication with the server.

10. Change the script to interpret the received msg, i.e., to get all attribute/value pairs in the message taking advantage of the '&' character used as delimiter.
    Consider using the string manipulation methods that exist in python. See this link for more information.
    TIP: Consider using the following methods: split.

11. Finally, you should get the respective values of each attribute in separate variables taking advantage of the use of the **'='** character used as a delimiter and print out the interpretation of each frame.

    **TIP:** Consider using the following methods: split.

```
Template to read and write text via console with ESP32:
/////////////////////////////////////////////////
// MECHAUZ 2023
// Rolando
/////////////////////////////////////////////////
int cnt = 0;
String str;
/////////////////////////////////////////////////
// Setup
/////////////////////////////////////////////////
void setup() {
 Serial.begin(9600);
 Serial.println("In setup function");
}
/////////////////////////////////////////////////
// Loop
/////////////////////////////////////////////////
void loop() {
 Serial.print("In loop function | cnt = ");
 Serial.println(cnt++);
 delay(1000);
 if (Serial.available() > 0) {
str = Serial.readString();
Serial.println(str.toInt());
Serial.println(str.toFloat());
}
}
```