1st Assignment

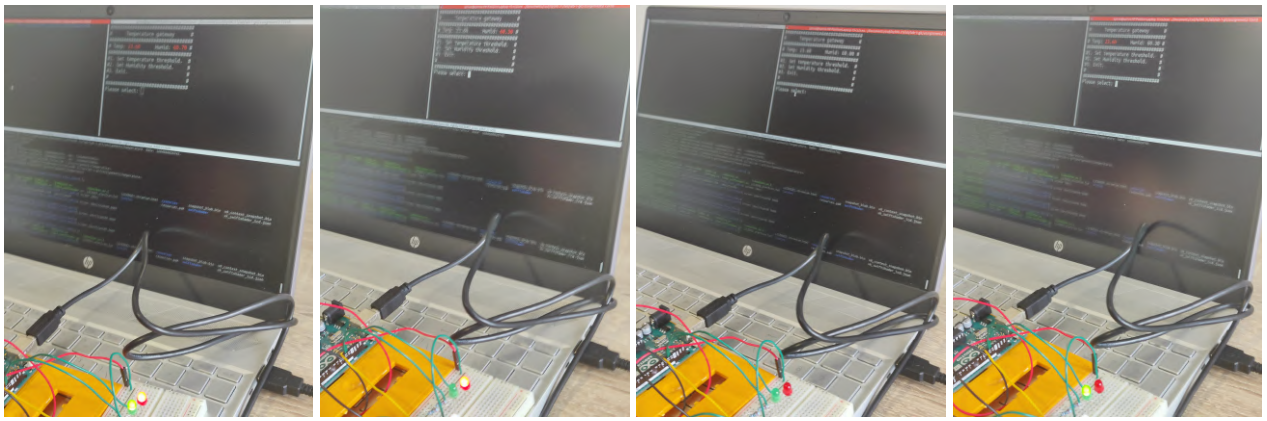
HY590.31 - IoT for Smart Cities

Vaggelis Gkolantas - csdp1329
Hilal Ozen - csdp1320
Spyros Tzagkarakis - csd4279

Contents

2 Data transmission over serial connection

Results



3 Data transmission over an IP-network

```
def readMeasurements():
    global lastUpdated
    global temperature
    global humidity

    # Checking if we need to update the measurements.
    if (lastUpdated is None or
    (lastUpdated is not None and int(time.time()-lastUpdated)>updateFrequency)):

        # Looping through all incommed measurements.
        # 12 bytes = [sign]XX.XX[sign]YY.YY where XX.XX is temperature and
        # YY.YY is the humidity.
        while ser.inWaiting() > 12:
            # Reading a measurement
            # Todo: read 12 bytes of data and store in a local variable. Use the
            #       ser variable.
            data_str = ser.read(12).decode("utf-8");
            temperature = float(data_str[0:6]);
            humidity = float(data_str[7:]);

            #print(temperature)
            #print(humidity)
            # Sets the values.
            # Todo: Extract the temperature and humidity from the data
            #       variable and store them in the global variables for
            #       temperature and humidity. Tip cast the extracted values
            #       to float.

        # Sending measurements to the server.
        sendMeasurements()

        # Updating the terminal.
        updateTerminalMetrics()

        # Updating the time variable.
        lastUpdated = time.time()
```

```
def sendToServer(request):
    # Creating a socket and connects to the set host and port.
    # Todo: Make a socket and connect to the host.

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.connect((host, port))

    # Dumps the request dictionary as a JSON object.
    # Sending the JSON message to the host. Note: The message must end with a |
    # character and the String should be encoded to bytes with .encode().
    # Todo: Create a JSON message and send it to the host with a trailing |.
    req_json = json.dumps(request);
    req_json = req_json + "|";
    sock.send(req_json.encode("utf-8"));

    # Reading input from the host.
    # Todo: Read the input from the host.
    data = sock.recv(2048);
    received = data.decode();

    # Closing the connection to the server.
    # Todo: Use the close() method on the socket.
    sock.close();

    received = received[:-1];

    # Creates a string from the receive bytes, removes the trailing | and parses
    # the string as a JSON dictionary.
    # Note the | must be removed from the string before the string is
    # loaded as a JSON dictionary.
    # Todo: Transform the recived bytes to a JSON dictionary.
    response = json.loads(received)

    return response;
```

Using these 2 functions we read our measurements from the serial port of the Arduino that has the sensor device attached to it and we send it to the server with ip 130.236.81.13

4 Hand-in questions

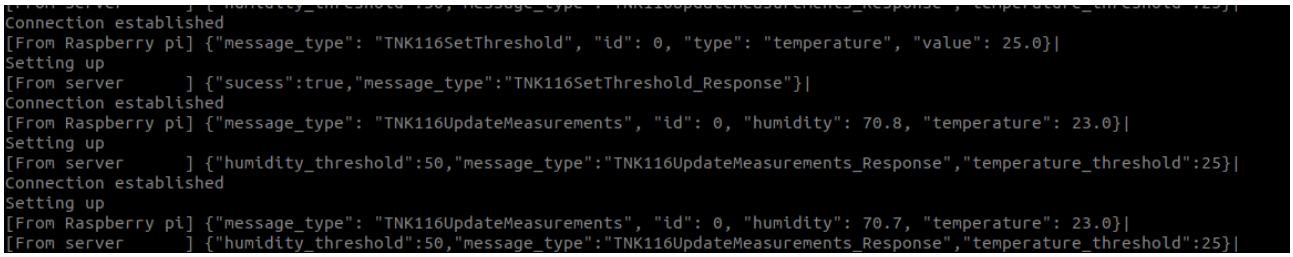
4.1 Sending data in a structured way

While sending the data as raw bytes of the float would result in a smaller data size (8 bytes compared to 48 bytes), it would also require additional effort to convert and interpret the raw bytes on the receiving end. The JSON message structure simplifies the process by providing a human-readable and easily parsable format, making it more convenient for data transmission and interpretation. Furthermore, if we have chosen to send the data with raw bytes, we would need to know in what order does the server wants to receive the data to interpret it correctly. By sending the JSON string, we don't care about the order because we include the type of information along with the information itself.

Thus, the decision to send the data as characters in a JSON format was likely made to prioritize ease of parsing, platform independence, and compatibility over the optimization of data size.

4.2 How is the data security on our traffic between the Raspberry Pi and the server

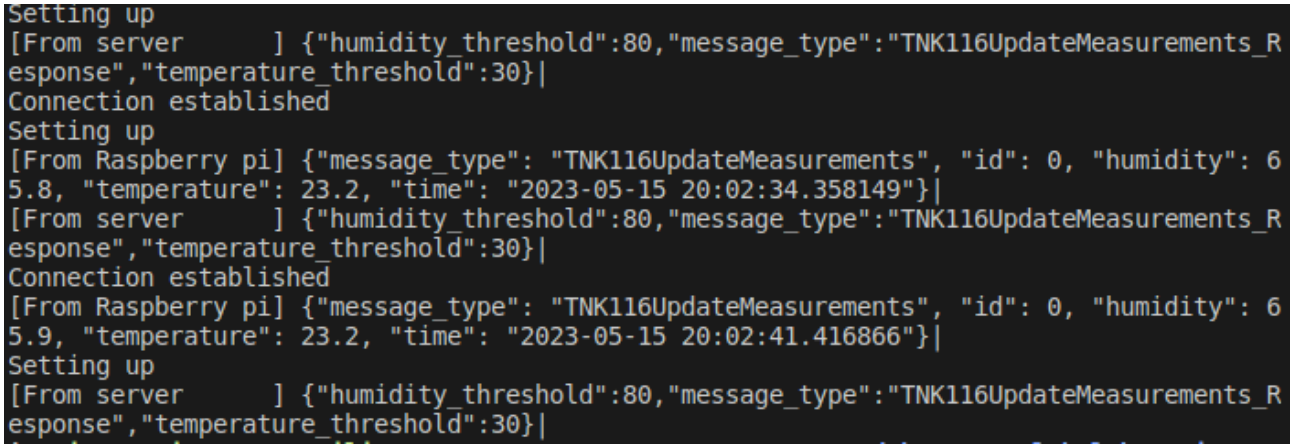
The traffic between the rasberry pi and the server is not ssl encrypted. A man-in-the-middle attack easily captures the communication between the devices. Similar attacks can also alter our messages, making our communication unreliable and or data security non existent. If our server is hooked with a database, then our database is at major risk as bad actors can flood it with fake data. Regarding security concerns of our data what we thought about is that a bad actor can analyze such data in order to predict apartment presence and thus plan physical theft.



proxy.py running on the Rasberry Pi

4.3 Timestamps

If the timestamps differ a lot then the information stored on the database will not be accurate. Times could overlap or be out of order making them invalid and/or hard to analyze in real-time. If our cloud server ran logic and had the ability to configure devices (such as changing thresholds) then this setup would not be reliable.



Adding the time field on the request and inspecting the request using the proxy