Design and Implementation of an Efficient Communication Protocol for IoT Meteorological Devices

Assignment for the course Computer and Communication Networks 2025/26

Context:

The company FIITMeteo develops and deploys meteorological IoT sensors intended for the long-term collection of meteorological data. Devices are battery-powered, and for this reason, the efficiency of transmitted data is a decisive factor. For these devices, it is necessary to prepare a proprietary communication protocol with which they will communicate with the server. It is also necessary to program the server itself together with a test program that will emulate individual devices.

Background Story:

Imagine that you are at the company FIITMeteo in the position of a junior developer. Since you have been in this position at the company for a little over a year, your senior developer (*your exercise instructor*) has decided to test you on a medium-difficulty task, your first independent project in which you will have to design a protocol and implement two programs according to the client's requirements. Since you have not yet done such a task, your senior arranged a meeting with you halfway through the project, where you need to demonstrate a PoC (Proof-of-Concept) of your solution. At this stage, the protocol should be designed and implemented using JSON structures (*First assignment submission*). After this meeting, you will start to optimize the protocol to binary; since these are battery-powered IoT devices, every byte saved in transmission extends the device's lifetime. As it usually happens, the client sometimes does not know exactly what he wants… you are sitting in the meeting room together with your senior, and just before you are about to hand over the project, the project manager (*the instructor responsible for the assignment*) calls you quickly to say that he forgot one change that was agreed with the client. This change is neither difficult nor large, but it is important for the client, and since you still have half an hour before the meeting, you have decided to incorporate this change (*additional implementation*). If you manage to incorporate this change, you will first go through the acceptance tests with the client, i.e., the tests that your solution must pass in order to be accepted by the client (*minimal requirements*). Then you will present the project to the client (final handover of the assignment to the instructor).

Before assigning this project to you, the project manager prepared a document based on meetings with the client, according to which you will have to execute the project. The document also contains Acceptance Tests. It also includes the grading system for the project that was agreed with the client (*scoring*). Your senior is transparent with you- if you do not

obtain at least half of the evaluation for the company (*obtaining half of the points at the first submission and obtaining half of the points at the final submission*), he will not be able to recommend you for a higher job position (*admission to the exam*).

Notes from the meeting with the client:

(requirements for the assignment)

The client has selected the following devices for his project:

1. **ThermoNode – 4 parameters**
   Focus: Temperature, humidity, and pressure
   • temperature: temperature in °C (−50.0 to +60.0, decimal number – 1 decimal place)
   • humidity: relative humidity in % (0.0–100.0, decimal number – 1 decimal place)
   • dew_point: dew point in °C (−50.0 to +60.0, decimal number – 1 decimal place)
   • pressure: atmospheric pressure in hPa (800.00–1100.00 [unclear], decimal number – 2 decimal places)

2. **WindSense – 4 parameters**
   Focus: Wind speed and characteristics
   • wind_speed: wind speed in m/s (0.0–50.0, decimal number – 1 decimal place)
   • wind_gust: wind gust in m/s (0.0–70.0, decimal number – 1 decimal place)
   • wind_direction: wind direction in degrees (0–359, integer)
   • turbulence: turbulence index (0.0–1.0, decimal number – 1 decimal place)

3. **RainDetect – 4 parameters**
   Focus: Precipitation and soil
   • rainfall: total precipitation over the last hour in mm (0.0–500.0, decimal number – 1 decimal place)
   • soil_moisture: soil moisture in % (0.0–100.0, decimal number – 1 decimal place)
   • flood_risk: flood risk level (0 – none, 1 – low, 2 – medium, 3 – high, 4 values)
   • rain_duration: duration of precipitation in minutes over the last hour (0–60, integer)

4. **AirQualityBox – 3 parameters**
   Focus: Air quality
   • co2: CO_2 concentration in ppm (300–5000, integer)
   • ozone: ozone concentration in µg/m³ (0.0–500.0, decimal number – 1 decimal place)
   • air_quality_index: AQI (0–500, integer)

Requirements:

P 1. Design and implementation of a proprietary protocol on top of the UDP protocol.

P 2. Two programs: a server that processes and displays data, and a test program that emulates all devices (one program – all devices).

P 3. Each sensor must request a token from the server via a registration message, which it must then include.

P 4. The test program must generate the messages and data that individual sensors send periodically, every 10 seconds; this functionality can be turned off and turned on again. Sending occurs in the background and a CLI is available with additional options.

P 5. Both programs must be implemented in Python; the permitted libraries are JSON, SOCKET, CRC, ASYNCIO, and THREADING (*with the instructor's approval, another language is also allowed*).

P 6. Ensuring the integrity of delivered data on the server; if the data are corrupted, the server requests them again.

P 7. Ability to introduce an error into the transmitted data for verification and requirement no. 6 in the test program.

P 8. Ability to send data entered by the user in the test program. Necessary to do for each sensor.

P 9. If any sensor is inactive, the server alerts the client by an output, periodically every 10 seconds, with the information format DISCONNECTED specified below. Based on the periodic message by which the server verifies whether the sensor is active, if within the time limit of 15 seconds no message arrives from the sensor *(that means the server waits for data from a registered sensor for 15 seconds; subsequently the server sends a verification message to the sensor; if within the limit it does not receive a message, then the DISCONNECTED output is activated if it does not receive a response to this message. If it receives a response, the information RECONNECTED is printed once).*

P 10. At any one time, only one sensor of each kind may be connected.

P 11. The output format must be adhered to. It is specified in the section Output Format.

P 12. IP addresses and communication ports can be set from the interface of both programs.

Based on the requirements, a list of recommended messages was prepared to meet these requirements; this list does not have to include all necessary messages:

- A message by which the device registers with the server; the response will be a message containing the device token. This token must be provided by the given device whenever it sends a message with data.
- A data message for each device.
- A message with which the server responds if the device token is invalid.
- A message with which the server confirms receipt of data if this message has reached the server.
- A message with which the server verifies that the given sensor is active, and the sensor's response to such a message.
- A message through which the server requests the latest message from the sensor if it has not been delivered.

Each of these messages must contain:

- Device type
- Timestamp in seconds (UNIX time)
- A flag if the device has a low battery
- The device token, which it obtains via the initialization message

Formatting of information texts:

Legend: CAPITAL_CASE – leave the text as it is , @ - ad specific value

- Sensor Registration:
  ,, INFO: @sensor_id REGISTERED at @timestamp \n."

- Received data message (text on server):

  ,,@timestamp - @sensor_id \n

name_of_param_1: @DATA_1; name_of_param_2: @DATA_2;... .... name_of_param_n: @DATA_n; \n “

- Received data message with LOW BATERY (text on server):

  ,,@timestamp - WARNING: LOW BATTERY @sensor_id \n

  name_of_param_1: @DATA_1; name_of_param_2: @DATA_2;... .... name_of_param_n: @DATA_n; \n “

- DISCONNECTED information (text on server):

  ,, WARNING: @sensor_id DISCONNECTED! \n “

- RECONNECTED information (text on server):

  ,, INFO: @sensor_id RECONNECTED! \n “

- Corrupted data (text on server)“
  ,, INFO: @sensor_id CORRUPTED DATA at @timestamp. REQUESTING DATA “

Acceptance Tests (UAT):

Testing conditions: communication must take place between two physical devices and must be monitored using the Wireshark program.

UAT 1: Pripojenie zariadení na server a automatické generovanie správ.	
Input conditions	*Server:* located in the menu. *Test program:* located in the menu, nothing has connected anywhere yet.
Test scenario	*1) Server:* in the menu, select the option to configure. Set IP and PORT. The program returns to the menu and the option to listen is selected. *2) Test program:* in the menu, select the option to configure. Set IP and PORT/PORTS. The program returns to the menu. *3) Test program:* select the option for automatic generation of data messages. The program returns to the menu.
Output conditions	*Server:* The server first prints information about the registration of individual sensors. Then, at regular intervals (approx. once every 10 seconds), information about a received message is displayed. (see the formatting above)
Test result	PASS/ACCEPTABLE/FAIL

UAT 2: Sending of user specified message	
Input conditions	*Server:* located in the menu. Devices are registered. *Test program:* located in the menu, regularly sends messages to the server.
Test scenario	*1)Test program:* in the menu, select the option to send a custom message. The program displays the names of the sensors along with the option to select this sensor. *2) Test program*: the user selects one of the devices. The program offers the user the option to notify the server of a low battery; the user selects this option. *3) Test program*: If there is a parameter that the user has not yet entered, it displays the name of the parameter and its possible values. The user enters a value from the parameter's range. If all values have been entered, proceed to step 4), otherwise repeat step 3). *4) Test program*: Sends a message with the user-entered data to the server.
Output conditions	*Server:* The server prints a data message for the case of a low battery voltage value. (see the formatting above)
Test result	PASS/ACCEPTABLE/FAIL

UAT 3: Introducing an error into the transmitted data.	
Input conditions	*Server:* located in the menu. Devices are registered. *Test program:* located in the menu, regularly sends messages to the server.
Test scenario	*1) Test program*: in the menu, select the option to introduce an error during sending. The program displays the names of the sensors along with the option to select one sensor. *2) Test program:* the user selects one of the devices. The message of this device is generated. However, during computation the checksum is intentionally corrupted. *3) Server*: Prints a message about receiving corrupted data. Subsequently, it requests these data from the device once again. *4) Test program:* After receiving the request to send the last data, it sends this data without introducing an error.
Output conditions	*Server:* prints the output for corrupted data, and then also prints the data message as soon as it has been delivered correctly (see the formatting above). It is necessary to show this process in Wireshark.
Test result	PASS/ACCEPTABLE/FAIL

UAT 4: Disconnection of the device and the mechanism for checking device activity.	
Input conditions	*Server:* located in the menu. Devices are registered. *Test program:* located in the menu, regularly sends messages to the server.
Test scenario	*1)Test program:* in the menu, select the option to check the sensor activity mechanism. The program displays the names of the sensors along with the option to select this sensor. *2) Test program:* the user selects one of the devices. Messages from this sensor stop being sent. *3) Server:* After the limit of 15 seconds from the last received message from the given device, it sends a verification message to the test program. If it does not receive a response within one second, it prints the DISCONNECTED information for the user. It then sends the verification message every 5 seconds, but at most 10 times. *4) Test program:* responds only after the server has attempted to contact it for the third time, and then continues as usual with automatic generation and sending of messages. The program returns to the main menu.
Output conditions	*Server:* Server prints the DISCONNECTED information twice, then prints the RECONNECTED information for the user. (see the format of informational outputs). It is necessary to show this process in Wireshark.
Test result	PASS/ACCEPTABLE/FAIL

UAT 5: Resending data if the server did not confirm that the message arrived.	
Input conditions	*Server:* located in the menu. Devices are registered. *Test program:* located in the menu, regularly sends messages to the server.
Test scenario	*1) Server:* in the menu, select the option “do not acknowledge the data message.” The program displays the names of the sensors along with the option to select this sensor. *2)Server:* the user selects one of the devices. For the data messages of this device, the server does not respond three times. *3)Test program*: after 1 second elapses, it sends the message again. This repeats until it receives a response from the server.
Output conditions	*Server:* prints the data message. (see the format of informational outputs), It is necessary to show this process in Wireshark..
Test result	PASS/ACCEPTABLE/FAIL

Proof of Concept (First submission):

Design and implementation of a JSON protocol. The JSON format was chosen because it is a human-readable form to simplify debugging and to verify the functionality of the individual programs, as well as how you have designed the protocol at the message level.

Scoring:

All items marked in red are mandatory at least in an acceptable form. At the same time, it applies that it is necessary to obtain a minimum of 2.5 points in total. This is so that we can accept an assignment that has minor deficiencies.

Part of Scoring	FAIL	Acceptable	PASS
UAT 1	0 b	0,5 b	1 b
UAT 2	0 b	0,5 b	1 b
UAT 3-5	0 b	0,5 b	1 b
UAT 3-5	0 b	0,5 b	1 b
Documentation	0 b	0,5 b	1 b
TOTAL	**0 b**	**2,5 b**	**5 b**

The documentation must at this stage contain at least the following particulars. At the same time, it must be submitted as a PDF according to the template you have available:

- Flowcharts with a description of the receive cycle and the send cycle in both programs. A brief textual description.
- Design of the individual messages in JSON, including an example for each message.
- Sequence diagram for communication for UAT1 and UAT2.
- List of libraries used.
- Design for measuring the efficiency of the transmitted data (do not forget to also account for the lower layers).

Outputs for submission:

A zip file named xsurname.zip (NOT RAR!) containing:

- The server program named server.xxx
- The test program named tester.xxxx
- The PDF documentation file named xsurname.pdf

Optimization to a binary protocol (Final submission):

An optimized protocol whose functionality is verified by two implemented programs. The goal is to represent the message type efficiently (e.g., 2 bits for 4 device types, or e.g., encode the timestamp as a 32-bit number if that is sufficient). Use the smallest possible number of bits for each parameter (e.g., if a parameter has a limited range and precision, choose an appropriate representation); likewise, make use of flags (single-bit parameters). Prepare a dissector for your protocol in Wireshark in the LUA language.

All items marked in red are mandatory at least in an acceptable form. At the same time, it applies that it is necessary to obtain a minimum of 7 points in total. This is so that we can accept an assignment that has minor deficiencies.

Part of scoring	**FAIL**	**Acceptable**	**PASS / Total Points**
UAT 1	0 b	1 b	**1,5 b**
UAT 2	0 b	1 b	**1,5 b**
UAT 3	0 b	1 b	**1,5 b**
UAT 4	0 b	1 b	**1,5 b**
UAT 5	0 b	1 b	**1,5 b**
LUA SCRIPT	0 b	**0,5 b - 1 b –** depending on cases the script covers	
Measurement and comparison of efficiency		**0,5 b - 1 b** – comparison of your own protocols, supported by a small table or chart.	**1,5b - 2 b** – in addition, comparison of the efficiency of your own protocol against at least one existing, practically used protocol, supported by a table or chart
Efficiency of data transmission	0 b - large data types used for small numbers, etc.	**0,5 b - 1,5 b** – appropriate use of data types, shown in Wireshark	**2 b** – highly efficient protocol, variable lengths as needed, etc.
Documentation		**0,5 b** – contains all required particulars	**1 b –** documentation from which it would be possible to implement the given programs and the protocol without additional questions
Understanding the topic in a broader context	**0 b – 0,5 b** – evaluation of the project, the protocols, and the measurement/benchmarking results themselves		
Additional implementation		**0,5 b**	**1 b**
TOTAL		**min 7 b**	**15 b**

The documentation has to contain all required particulars. At the same time, it must be submitted as a PDF according to the template you have available:

- Flowcharts with a description of the receive cycle and the send cycle in both programs. A brief textual description.
- Design of the individual messages in JSON, including an example for each message.
- Structures of binary headers and fields.
- Sequence diagram for communication for UAT1–5.
- List of libraries used.
- Example of encapsulation of a message down to the L2 level.
- Measurement of the efficiency of the transmitted data (do not forget to also account for the lower layers).
- Comparison of the efficiency of transmitted data between your own protocols.
- Appendix A (around 150 words) – Evaluation of the educational aspect of the project: What I learned thanks to this assignment. How these skills can help me in further studies and in the job market.

Outputs:

A zip file named xsurname.zip (NOT RAR!) containing:

- The server program named server.xxx
- The test program named tester.xxxx
- The PDF documentation file named xsurname.pdf