IoT: Design Network Protocol

Using UDP sockets, you are to write a client and server program that allows the client to perform a set of operations on a server (light bulb)

The client will perform the following functions:

1. Read in 2 arguments from the command line (to connect to the server [light bulb])
   1. IP address of the server (e.g. 127.0.0.1).
   2. Port of the server (e.g. 10000).
2. Send a request in order to turn the light bulb on or off and the functions shown in the request format
   1. If a response occurs within the timeout period, print the server’s response as shown below.
   2. If there is no response, re-send the message (same sequence number) for a max of 3 times before printing an error message and exiting.

The server will perform the following functions:

1. Read in 2 arguments from the command line:
   1. IP address of the server (e.g. 127.0.0.1).
   2. Port of the server (e.g. 10000).
2. Prepare to receive a response from the client.
3. Read the message and perform the required functions:
   1. Turn bulb on/off
   2. Change color
   3. Dim light
4. Respond to the client using the response format shown below.
5. If the request is incomplete, then send a default message.
6. Send an error message if:
   1. Bulb is not working
   2. Color specified is not supported
   3. Request format is not recognized

**Simplifying Assumptions:**

* Set the lightbulb off by default.
* If you get a response from the server, print: “Lightbulb is on, color: [specified color]”.
* If you didn’t receive a response from the server, print: “Lightbulb is off, no color”.
* Allow to send the request for a max of 3 times, if you didn’t receive a response from the server.
* To turn the lightbulb, the color must be set to black.
* Allow the request to support any color.
* One light bulb can be controlled at a time.
* There can only be one answer in the answer section.

The **request** has the following format:

0 1 2 3 4 (bytes)

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Message Type (1) | Return Code (0) |

-------------------------------------------------------------------------------------

| Message Identifier (e.g. 5) |

-------------------------------------------------------------------------------------

| Turn (0 or 1) | Dim light (e.g. 0.3) |

-------------------------------------------------------------------------------------

| Color (e.g. “#5dbcd2”, “rgb(93, 188, 210)” ) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

The **response** has the following format

0 1 2 3 4 (bytes)

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Message Type (1) | Return Code (0) |

-------------------------------------------------------------------------------------

| Message Identifier (e.g. 5) |

-------------------------------------------------------------------------------------

| Turn (0 or 1) | Dim light (e.g. 0.3) |

-------------------------------------------------------------------------------------

| Color (e.g. “Light-Blue”) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

* Message Type (16 bits): 1 on Request; 2 on Response.
* Return Code (16 bits): 0 if error; 1 if no error.
* Message Identifier: Uniquely identify a message in a request, server echoes the same number back in response. It should be randomly generated between 1 and 100.
* Power: 0 – off; 1 – on.
* Dim: Set the dim to the default value if power is 0. Dim is set to the given value if power is 1.
* If the power is 0, dim is set to a default value of 0; if power is 1, dim is set to the given value.
* Color: Only one color can be set in a single request. Color should be implemented in 2 ways: Hex code or RGB code. Hex code must always start with ‘#’ and RBG code should contain three values (0 – 255) separated by commas and enclosed in parentheses. See the request format above for an example. If code is illegal send an error message.