**Internet of Things Assignment: Light Bulb**

Here, we are implementing a protocol that communicates over UDP with an Internet of Things device, a light bulb. The client will send a request to the server to turn on/off the light bulb, check if it’s on and what color it is currently set to, and to change the color.

The client will perform the following functions:

1. Read in 3 or 4 arguments from the command line:
   1. IP address of server (127.0.0.1)
   2. Port of server (e.g. 9999)
   3. The function
   4. The color
2. Send a request with the specified function to the server using the message format specified. There should be a number associated with each function. How you organize that is up to you.
3. Wait for a response using a 1 second timeout period.
   1. If a response arrives within the timeout period, print out the server response as shown in this document
   2. If not, re-send the message (same sequence number) for a maximum of 3 attempts before printing an applicable message and exiting

The server will perform the following functions:

1. Read in 2 arguments from the command line:
   1. IP address of server (127.0.0.1)
   2. Port of server (e.g. 9999)
2. Server should print that it’s running.
3. Create an array of tuples, or a map, in which each number is matched up to its respective function and color.
4. Perform the function specified, utilizing the aforementioned array
5. Respond to requests from the client using the message format specified
6. Return an error if the bulb isn’t functioning (create this intentionally by using a random number and an if statement), or if the color doesn’t exist (not in the array).

Simplifying Assumptions:

* Only one question in question section
* Only one answer in answer section

In a request, the application data has the following format in network byte order:

0 1 2 3 4 (bytes)

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

| Message Type (1) | Return Code (0) | ----------------------------------------------------------------------------------- | Message Identifier (e.g. 5) |

------------------------------------------------------------------------------------- | Question Length (e.g 1 or 2) | Answer Length (0) |

------------------------------------------------------------------------------------ // Question String (e.g. ‘0’ which could be to give the status) //

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

In a response, the application data has the following format in network byte order:

0 1 2 3 4 (bytes)

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

| Message Type (2) | Return Code (0,1,2 or 3) |

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

| Message Identifier (e.g. 5) |

------------------------------------------------------------------------------------- | Question Length (e.g 1 or 2) | Answer Length (41) |

------------------------------------------------------------------------------------ | Question Section (e.g. ‘0’) //

// |

-------------------------------------------------------------------------------------- | Answer Section //

// (e.g. ‘Light on. Color red.’) |

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

Message Type (16 bits): 1 on request; 2 on response

Return Code (16 bits) : 0 on request ; in response, 0 if function was performed, 1 if the color wasn’t supported,2 if the bulb wasn’t functioning, 3 if the function isn’t supported.

Message Identifier (32 bits): Uniquely identifies a message in a request, server echoes same number back in response. Should be generated randomly in range between 1 and 100

Question Length (16 bits): In request and response, length of resource record in Question Section in bytes.

Answer Length (16 bits): 0 in request (because no answer section). In response, length of resource record in Answer Section in bytes.

Question Section (16 bits): In request, you can send 1 or 2 numbers depending on the function. Echoed back in server response

Answer Section (variable length): In request, there is no answer section. In response, the server states the current status of the light bulb. (e.g. ‘Light bulb on. Color red’). Empty if color not supported.

**Test Output**

1. Test Case 1: Client Output Example (Function supported):

$ python3 client.py 127.0.0.1 9999 On

Sending Request to 127.0.0.1, 9999

Received Response from 127.0.0.1, 9999

Answer: Light Bulb on. Color red.

2. Test Case 2: Client Output Example (Function empty or not supported)

$ python3 client.py 127.0.0.1 9999 Break

Sending Request to 127.0.0.1, 9999

Error: Function not supported.

3. Test Case 3: Client Output Example (Color not supported)

$ python 3 client.py 127.0.0.1 9999 Violet

Sending Request to 127.0.0.1, 9999

Received Response from 127.0.0.1

Answer: Color not supported.

4. Test Case 3: Client Output Example (Server does not respond)

$ python3 client.py 127.0.0.1 9999 Off

Sending Request to 127.0.0.1, 9999:

Request timed out …

Sending Request to 127.0.0.1, 9999:

Request timed out …

Sending Request to 127.0.0.1, 9999:

Request timed out … Exiting Program.