Programming Assignment: Communication with IoT Device

Using UDP sockets, you will write a simplified version of a RGB light hub client and server. The server will be responsible for setting which light to a specific color. The client will send a request to the server to change the color of a particular light, and the server will change the specified light to the color passed from the client. The client and server will communicate using the message format specified in this document.

**The client will perform the following functions:**

1. Read in 3 arguments from the command line:
   1. IP address of server (127.0.0.1)
   2. Port of server (e.g. 9999)
   3. Color light to be changed to or status
      1. Stat (for current status of light)
      2. Red
      3. Blue
      4. Off
2. Send a request with the color to the server using the message format specified
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 3 arguments from the command line:
   1. IP address of server (127.0.0.1)
   2. Pert of server(e.g. 9999)
   3. Status / color of light (same as client) or can be broken
2. Initialize the light with a color(e.g. Light is red)
3. Respond to requests from the client by changing the color of light specified
4. Return an error if the light color does not exist or light is broken
5. Once a light is changed, print out color and status
6. Return code is 0 if working properly, 1 if the light does not exist, and 2 if light is broken

**Test Cases:**

1. Change existing light to another color
2. Change non-existing light or non-existing color
3. Change color when server is not running (On Windows, run the server but command out responses)

**Message Format:**

In the **reques**t, the application data has the following format:

0 1 2 3 4 (bytes)

==============================================

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

==============================================

| Message Identifier (e.g. 5) |

==============================================

| Color (e.g. Red) |

==============================================

In the **response**, the application data has the following format:

0 1 2 3 4 (bytes)

==============================================

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

==============================================

| Message Identifier (e.g. 5) |

==============================================

| Color (e.g. Red) |

==============================================

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

Return Code (16 bits): 0 on request; in response, 0 if light changed, 1 if light does not exist, 2 if color does not exist

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

Light number (16 bits): number of the light that is to be changed

Color (16 bits): the color that the light is to be changed into

Test output working:

server.py 127.0.0.1 12000 off (this should initialize a light that is off)

Sending data to server...

Client.py 127.0.0.1 12000 stat (get status of light)

Sending response to client...

Light is Off (client should print out that the light is off)

Test output changing lights:

server.py 127.0.0.1 12000 off (this should initialize a light that is off)

Sending data to server...

Client.py 127.0.0.1 12000 red (get status of light)

Sending response to client...

Light has been changed to red (client should print out the color of light)

Test output timed out:

server.py 127.0.0.1 12000 off (this should initialize a light that is off)

Sending data to server...

Timed out

Sending data to server…

Timed out

Sending data to server…

Timed out

Server did not respond (print that server has not responded)