Program 2 - Simple Protocol

On this page

~~**This is a GROUP Assignment**~~

Objectives

1. Learn to create network packets.
2. Learn how packets can be sent over the network.
3. Familiarize you with the concept of sockets.
4. Learn packing structures, endianness, unpacking, and interpreting network data.
5. Learn how to use actual data from a packet.
6. Use packet capture to visually inspect protocols.

Overview

In this warmup project, you are going to implement a client and a server program that will send a command over the network. The command simply turns on an LED light. However, the LED light speaks a special protocol. All we know about this protocol is the packet format used for turning on the light. Your task is to create this packet and then send it to the server for turning on this light.

Server Specifications

The server takes two arguments:

```
$ lightserver -p <PORT> -l <LOG FILE LOCATION>
```

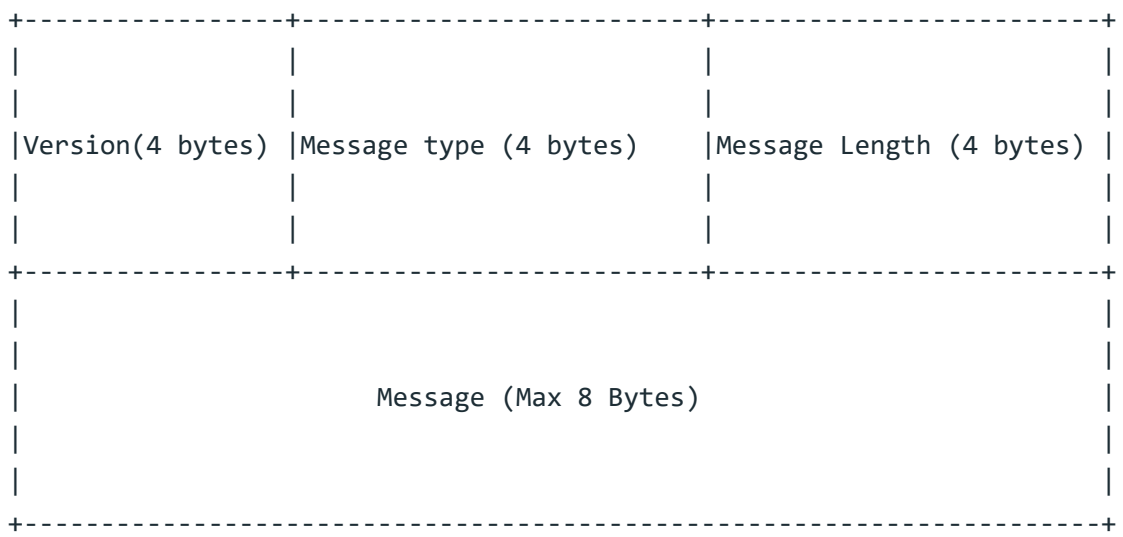
1. `PORT` - The port server listens on.

2. `Log file location` - Where you will keep a record of actions.

```
For example:
$ lightserver -p 30000 -l /tmp/logfile
```

Deliverables (each worth *5* points)

1. Server must parse two command line arguments, port and log locations.
2. The server must *not* exit after receiving a single packet.
3. Once a client connects, it logs a message in the following format `"Received connection from <CLIENT IP, PORT> "`
4. Once it receives a hello message from the client, it logs the connection and sends a hello back to the client.
5. You can assume the packet format is the following:



6. It receives the packet header first, followed by the message. *Hint: You need two RECV calls*.
7. Check if Version == 17. If not, log an error message `VERSION MISMATCH` and continue to listen. Do not exit.

8. If Version == 17, check the message type. If message Type is 1 - the corresponding command is `LIGHTON`. If message type is 2 - the corresponding command is `LIGHTOFF`. No other command is supported.
9. If the server sees a supported command, log "EXECUTING SUPPORTED COMMAND: COMMANDNAME", else log <"IGNORING UNKNOWN COMMAND: COMMANDNAME".
10. Send back a "SUCCESS" message to the client.
11. It turns on or turns off the LED on your circuit based on the command sent by the client.
12. Make sure server does not exit on 0 byte messages.

Client Specifications

```
$ lightclient -s <SERVER-IP> -p <PORT> -l LOGFILE
```

The client takes three arguments:

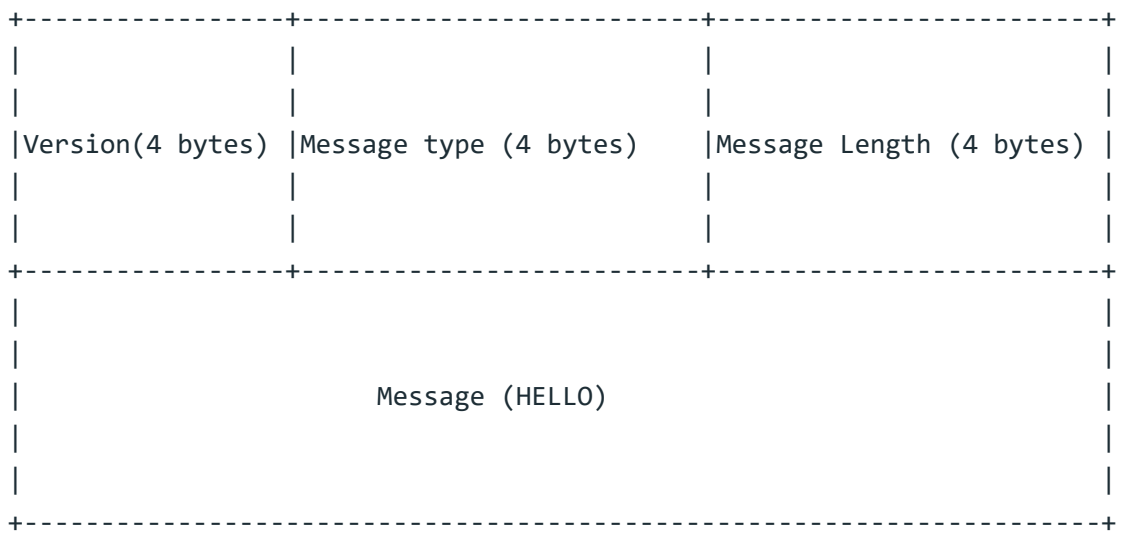
1. `Server IP` - The IP address of the server.
2. `PORT` - The port the server listens on.
3. `Log file location` - Where you will keep a record of packets you received.

```
For example:
$ lightclient -s 192.168.2.1 -p 6543 -l LOGFILE
```

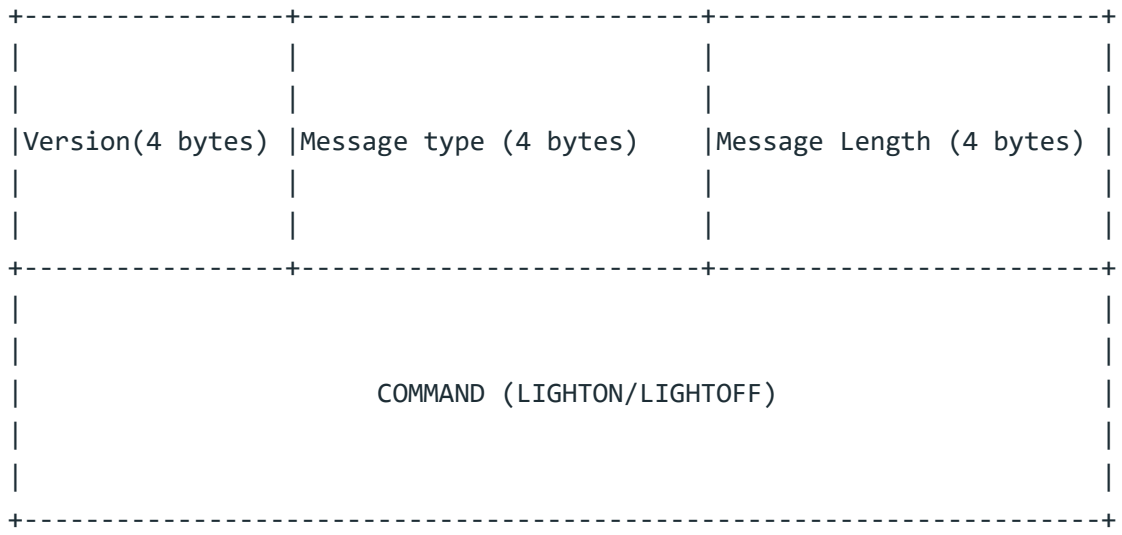
Client Requirements (each worth 5 points, item 7 is worth 10 points)

1. The client must parse three command line arguments, server, port, and logfile.
2. The client should connect to the server on the specified port.

3. Constructs and sends a hello packet to the server.



4. Receive reply from Server - if version is 17, log “VERSION ACCEPTED”, else log - “VERSION MISMATCH”
5. If version is accepted, send a command packet.



6. Receive the server’s reply, log the reply, and gracefully shutdown the socket. You can assume the server always replies with a “SUCCESS” message for this assignment.
7. Use TCPDUMP or Wireshark to capture the interactions, turn the .pcap file in with the

assignment.

Additional requirements:

1. Code must compile/run on your VM. You will demo this to the TA.
2. You must pack the packet in a structure. If you are using python, use the struct module.
3. Pay extra attention to byte-order encoding before sending the packet. Big-endianness is the dominant ordering in today's network protocols.

Sample Output

Server side

```
Received connection from (IP, PORT):  ('127.0.0.1', 53888)
Received Data: version: 17 message_type: 1 length: 1280
VERSION ACCEPTED
Received Data: version: 17 message_type: 2 length: 1792
VERSION ACCEPTED
EXECUTING SUPPORTED COMMAND:  LIGHTON
Returning SUCCESS
Received connection from (IP, PORT):  ('127.0.0.1', 53890)
Received Data: version: 17 message_type: 1 length: 1280
VERSION ACCEPTED
Received Data: version: 17 message_type: 2 length: 1792
VERSION ACCEPTED
EXECUTING SUPPORTED COMMAND:  LIGHTON
Returning SUCCESS
```

Client Side

Run 1

```
Received Data: version: 17 type: 1 length: 1280
```

```
VERSION ACCEPTED
Received Message Hello
Sending command
Received Data: version: 17 type: 2 length: 1792
VERSION ACCEPTED
Received Message SUCCESS
Command Successful
Closing socket
```

Run 2

```
Sending HELLO Packet
Received Data: version: 17 type: 1 length: 1280
VERSION ACCEPTED
Received Message Hello
Sending command
Received Data: version: 17 type: 2 length: 1792
VERSION ACCEPTED
Received Message SUCCESS
Command Successful
Closing socket
```