WESTERN UNIVERSITY - CANADA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SE 3314B – COMPUTER NETWORKS APPLICATIONS

Assignment 1: Client/Server Application

Due Date: March 3, 2023

This assignment is to be done individually. Cheating is prohibited in this assignment, that is not to show your code to any other student, do not look at any other student's code, and do not put your code in any public domain. However, getting help is easy and available, just ask the instructor or the TAs. At the same time, you are encouraged to discuss your problems (if any) with other students in general terms, but the actual program design and code must be your own.

Please note that the consequences of cheating are much, much worse than getting a low mark for that particular item of work. The very best case is that you get a zero for the whole thing. Keep in mind that turning in your own work would have at least gotten you a few marks, better than a zero.

Code submissions will be graded manually, and the code similarity check will be made using the similarity-detection software system.

1. Objectives

This assignment is a refresher for socket programming. I assume you have done some socket programming prior to this course, either in ECE4436a or equivalent experience. If your socket programming skill is rusty or shaky or if you're still unsure how to build a socket program using Node.js environment, this assignment gives you a chance to shore up both.

You are given a JavaScript code skeleton including some useful methods by which you will build a simple JavaScript based client-server remote image viewer. When completed, the server will provide an image to the clients to download and then display it directly using the default image viewer supported by the operating system. This assignment should provide you with direct feedback on its fitness: a correctly completed codes will display a recognizable image on the client. Unfinished or incorrectly written codes probably will not. We will use this remote image display tool throughout the term to help you visualize the effects of various p2p network protocols.

2. Installing Node.js

The first thing to do before you begin is install the JavaScript compiler (the Node.js platform). If you have done so already, please skip this section. Node can be downloaded from the project home page: http://nodejs.org/ where you will be presented with a package installer for your operating system. Binary installers are available for Unix, Mac, and Windows. The stable release version of Node at the time of this assignment is 18.14.0 (LTS). Node provides a JavaScript runtime environment, which you can access at any time by going into your command prompt or terminal window and typing node. For the purposes of the assignment you will be running your main program (JavaScript source file) rather than typing code directly into Node. To run a JavaScript file, you run the node command with a parameter like this: **node filename.js**.

3. Importing third-party modules

Node ships with a **package management utility** called **npm**, which enables you to import third-party libraries into your workspace to use in your code. Package management is an important aspect of working with Node; without it, you would have to program all of your applications from the ground up, reinventing solutions to common problems that have already been solved (and shared) by hundreds of other developers.

To install these third-party modules (like open package) using npm, run the npm command with the module name as its parameter, like this: **npm install open --save.**

As you develop Node.js applications, you will undoubtedly make use of modules. In order to install all your application modules automatically when being deployed in the running and evaluation device, you need to use package.json file concept.

4. Task 1: The Images Management Server

You are required to develop an asynchronous images management server that can handle each client's requests individually and at the same time it services many (hundreds or thousands) of clients.

This means, as each client connects to the server port and host, the main program will fire a pre-defined **Callback** function asynchronously. Because the **server.listen()** method creates a new **Socket** object for the connection, the original Socket object will be free to accept more client connections. On the command line, the server is run simply by typing:

```
> node ImageDB
```

The server does not have any required command-line option. Upon start up, the server initializes a TCP socket and obtain an ephemeral port number from the kernel which it prints out to the console. The server will also start a timer to tick every 10 milliseconds, that is initialized by a random value from 1 to 999 and keep incremented by 1 every tick along the lifetime of the server execution. Then it waits for a query message from the clients, loads the requested image from a set of available images exits in the same folder/directory (it is launched from) and transfers the image to the client. Note that, the timer resets when reaches on the value 2^{32}. Your software application in this task should include, but not limited to, the following components.

4.1. Console Interface

Node.js command console will be an acceptable UI for your application in this assignment. Your application will display the core operations of the server into sequential and scrolling up form of interaction. Figure 1 shows a sample look of your server interface.

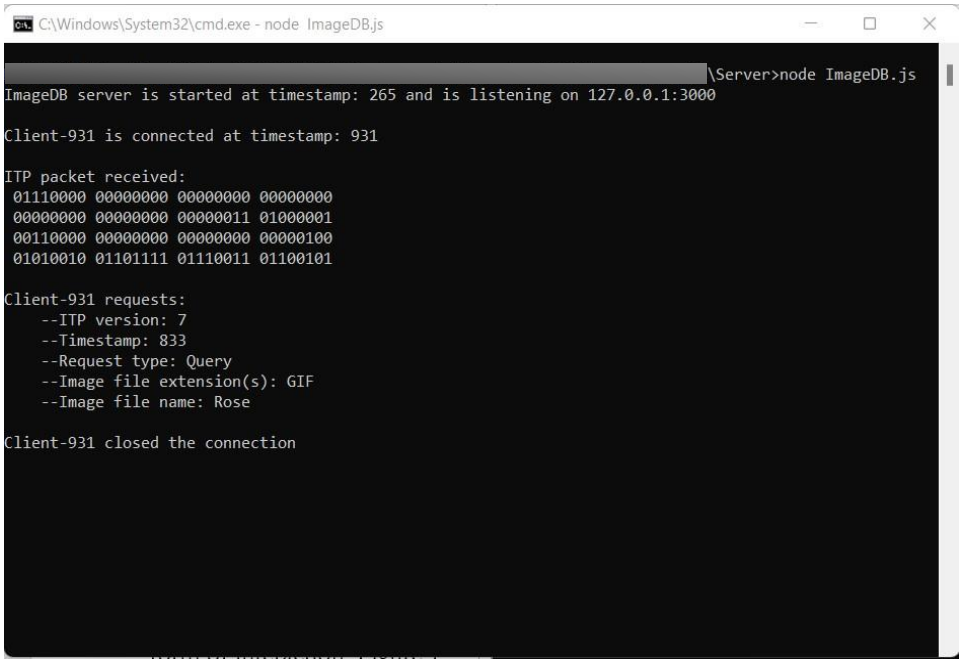


Figure 1: Sample look of your server interface

4.2. Response packetization

Module – ITPResponse.js

The image management server uses an SE3314b' custom Image Transport Protocol (ITP) for image files delivery. The core server functionality is to maintain the client's connection, send and receive the ITP packets (i.e., the encapsulated ITP packets). However, the main functionality of this module is to maintain the ITP packetization process, i.e., to form the ITP response packet. The size of ITP response packet **header** is 12 and only 12 bytes and should be sent in specific **bits** order as shown in Figure 2.

It is recommended to use the additional methods given in provided code to form the header bit stream.

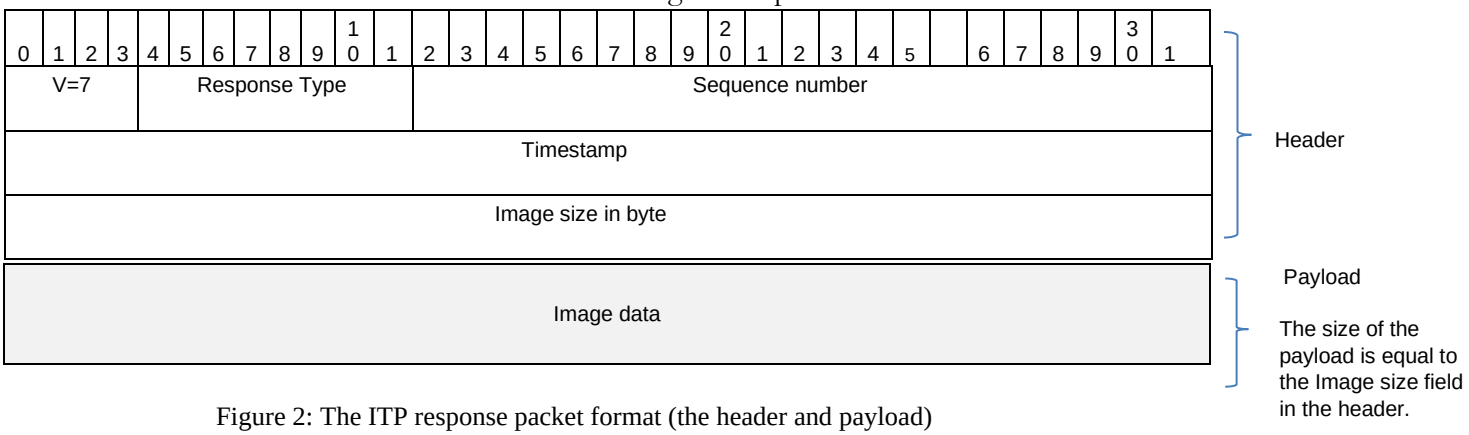


Figure 2: The ITP response packet format (the header and payload)

Here are the descriptions of the ITP response packet fields:

- (V) is a 4-bit ITP version field. You must set this to 7. The bits order should be as follows:

Bit position	0	1	2	3
Bit value	0	1	1	1

- The response type is 8-bit value so that the value 0 means "Query", 1 means "Found", 2 means "Not found", and 3 means "Busy".
- Set the sequence number. The sequence number of the first packet is chosen randomly; it is incremented by 1 for each subsequent packet sent out of the server. This field is 20 bits in length and the increment will be performed mod 2^{20}.
- The timestamp is a 32-bit field has the current value of the server's timer.
- The image size is a 32-bit field which we will need to extract the image data from the packet and display it.
- The actual image data (part of the payload) will then follow the image file name field.

4.3. Clients Handler Module – ClientsHandler.js

This module requires the Net package to provide all the functionality to create the server TCP socket, listen to client connections, receive clients' commands, and send the corresponding responses.

When the module receives a query packet, it checks that the packet has the expected version number and request type is "Query", and that the queried images (or some of them) are existing. The definition of the query packet is shown in Section 5.2 below.

If none of the requested images exists, it sends only the completed ITP packet (no payload) with response type "Not found". If the image files (or some of them) exist, it sends the ITP packet and the payload(s) to the client. You must fill in all the contents of the packet.

4.4. Main Application – ImageDB.js

The main application of the server requires the above modules (ITPpacket.js, ClientsHander.js, and other like fs, etc.) in order to function. The basic functionality of the main application is to accept connections, receive client command and respond accordingly.

It is also responsible to display (log) into the console all information about the connected/disconnected clients, and client's requests (ITP request packet bits stream along with the fields data) as shown in Figure 1.

5. Task 2: The Client Side

Your second task is to write the client code. The client takes a single line command with the following format:

```
> node GetImage -s <serverIP>:<port> -q <image name> -v <version>
```

where '>' indicates the console prompt, which you don't type in. The -s option tells **GetImage** which server to connect to. You must provide the IP address of the server, along with the port number the server is listening on, separated by a colon. The angle brackets ("< >") indicate that you are to provide the actual values for the required arguments. You don't enter the angle brackets themselves when you run the program. The q option tells the **GetImage** program which image files to query the server for. You replace "<image name>" with the image file name including image type, again you don't enter the angle brackets themselves. The -v option allows you to change the version number of the ITP packet sent. You should use it to test whether your server function is checking the version field of all incoming ITP packets correctly. In this assignment, 7 is the supported version of the ITP.

Your software application in this task should include, but not limited to, the following components.

5.1. Console Interface

Your application will display the core operations of the client side into sequential and scrolling up form of interaction. Figure 3 shows a sample look of your client interface.

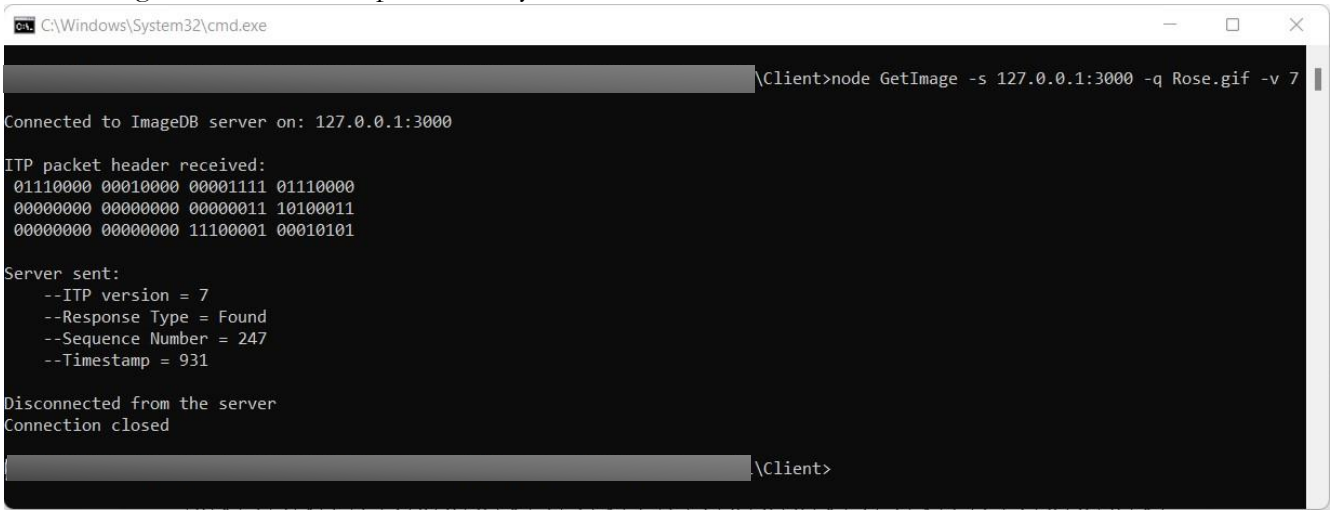


Figure 3: Sample look of your client interface

5.2. Request packetization Module – ITPRequest.js

As you already know, our image management server uses an SE3314b custom Image Transport Protocol (ITP) for image files delivery. The main functionality of this module is to maintain the ITP packetization process, i.e., to form the ITP request packet. The ITP request packet is 12 bytes header followed by the payload (the image file name). The ITP request packet should be sent in a specific structure as shown in Figure 4.

It is recommended to use the additional methods given in provided code to form the header bit stream.

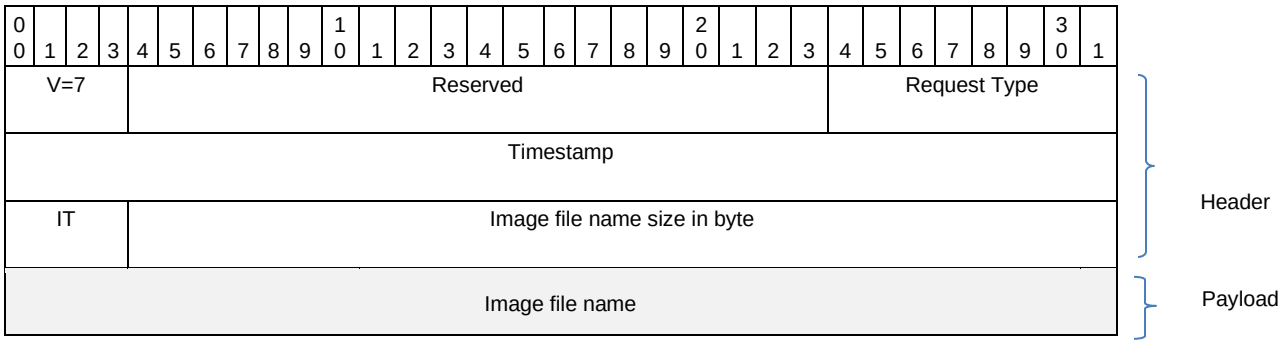


Figure 4: The ITP request packet format

Here are the descriptions of the ITP request packet fields:

- (V) is a 4-bit ITP version field. You must set this to 7.
- Reserved is a 20-bit field, not used in this assignment.
- The request type is 8-bit field. In this assignment it is always set by the value 0 and means 'Query'.
- The timestamp is a 32-bit field has the current value of the client's timer.
- (IT) Image Type is a 4-bit field indicates the type of the images as follows: 1 - BMP, 2 - JPEG, 3 - GIF, 4 - PNG, 5 - TIFF, and 15 – RAW.
- The file name size is a 28-bit field hold the number of bytes needed to store the image file name.
- The image file name, without the file extension is a field to store the name of the requested image. The size of this field is equal to the length of the image file name.

5.3. Main Application – GetImage.js

This module requires the Net module to provide all the functionality to create the client TCP socket, to connect to the server, send queries, and receive responses.

The first task of this module is to connect to the server using the IPv4 address and post number given in the command line arguments. At the same time the client connects to the server it sends the query packet that include the required image file name to the server using the packet structure described in Figure 4. This module will also receive the return packet from the server. The return packet structure must be of the form as shown in Figure 2 above. Once this module received a correct ITP packet format, it extracts the images contents from the packet and saved it in the client current folder. Finally, it calls the default image viewer in client's computer to display the downloaded images. You may need to use the third-party NPM package called 'open'.

6. Testing Your Code

You can develop your code on either Linux, Mac OS X, or Windows platform, as Node.js is available for all these platforms and the code you develop on one platform can run directly on the others. The easiest way to test your code is to run both the server and client on your local host, e.g., your laptop. After completing the code as described above, you can run it as follows.

- First, start the server as shown below and recognize the port number and server address on which the server is accepting client connections.

```
> node ImageDB

   ImageDB server is started at timestamp: 265 and is listening on 127.0.0.1:3000
```

- Then start the client program 'GetImage', as shown below. Enter the recognized server port number and server address, include image file names you need to request.

```
> node GetImage -s 127.0.0.1:3000 -q Rose.gif -v 7
Connected to ImageDB server on: 127.0.0.1:3000

ITP packet header received:
 01110000 00010000 00001111 01110000
 00000000 00000000 00000011 10100011
 00000000 00000000 11100001 00010101

Server sent:
    --ITP version = 7
    --Response Type = Found
    --Sequence Number = 247
    --Timestamp = 931

Disconnected from the server
Connection closed

>
```

- If the above command entered correctly, your server should send the response packet holding the image data and then the client will parse this data packet extract the image data and display it in the screen, as shown in Figure 5.

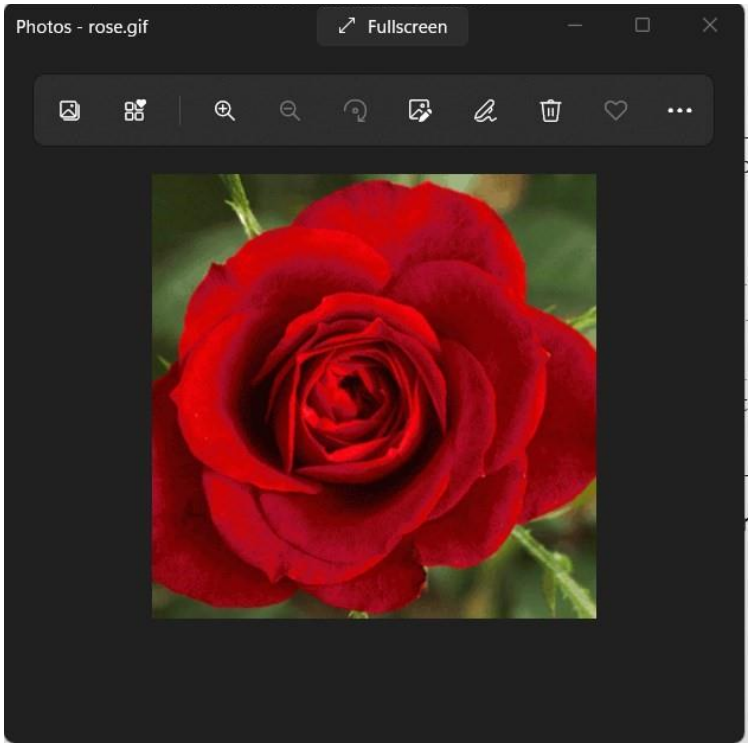


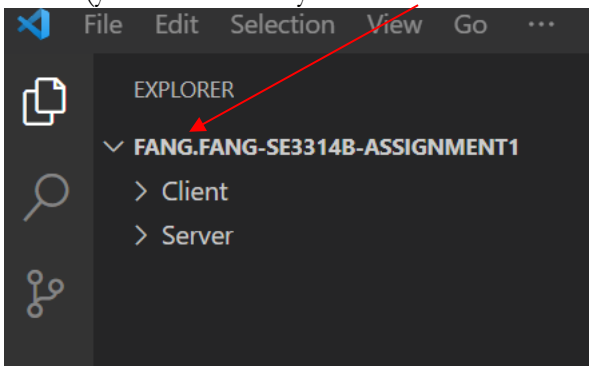
Figure 5

- The server should accept the client request and update its screen as shown below. Note that, the server is always running accepting requests from many clients.

```
> node ImageDB
ImageDB server is started at timestamp: 265 and is listening on 127.0.0.1:3000
Client-931 is connected at timestamp: 931

ITP packet received:
 01110000 00000000 00000000 00000000
 00000000 00000000 00000011 01000001
 00110000 00000000 00000000 00000100
 01010010 01101111 01110011 01100101

Client-931 requests:
    --ITP version: 7
    --Timestamp: 833
    --Request type: Query
    --Image file extension(s): GIF
    --Image file name: Rose

Client-931 closed the connection
```

Hand In

- Your source code should be fully commented and formatted.
- Your project folder structure should be created such that all code for the server modules should be under separate folder named 'Server' and all code related to the client program should be saved in a folder named 'Client, as follows: (you need to use your *uwoID* instead of 'fang.fang').



- Compress these two folders in an achieve file (zip file) and name it ***yourUWOID-SE3314bassignment1.zip***. This should include all JS files for the server and the client including the **package.json** file.
- Submit this zip file on OWL by the due date mentioned above.