**University of Plymouth**

**NET107:** Principles of Infrastructure

Coursework

Traffic Lights – Report

*Goel Biju* (10576090), *S. Tajwar* (10584193)

School of Computing, Electronics and Mathematics

Academic Year: **2017-18**

May 21, 2018

Table of Contents

[Overview 1](#_Toc514664888)

[Client & Server 1](#_Toc514664889)

[Usage of Server & Client 2](#_Toc514664890)

[Application Layer Protocol 5](#_Toc514664891)

[Client 5](#_Toc514664892)

[Server 6](#_Toc514664893)

[Evaluation 7](#_Toc514664894)

[Further Development 7](#_Toc514664895)

[Contributions 8](#_Toc514664896)

# Overview

The Traffic light client and server aims to simulate a crossroad of cars. By the process of client to server interaction, events are sequenced so that clients can communicate to the server to notify when a light change is necessary; in this scenario when there are 10 or more cars. The traffic light server handles multiple connections (with 4 clients connecting to a single server to form a virtual crossroad) from different IP addresses.

Using each connection, information exchange occurs between both client and server to help the server control traffic flow on the crossroad. An application layer protocol was devised to achieve a method of communication between a single client and server.

# Client & Server

Each client represents one road in the crossroad. The number of cars being sent from the client are fetched up in the server and thus acts upon the request of the client which is basically deciding when the cars should be allowed to go and on which side of the crossroad.

The user must make sure that the server and the client relates to the help of IP addresses.

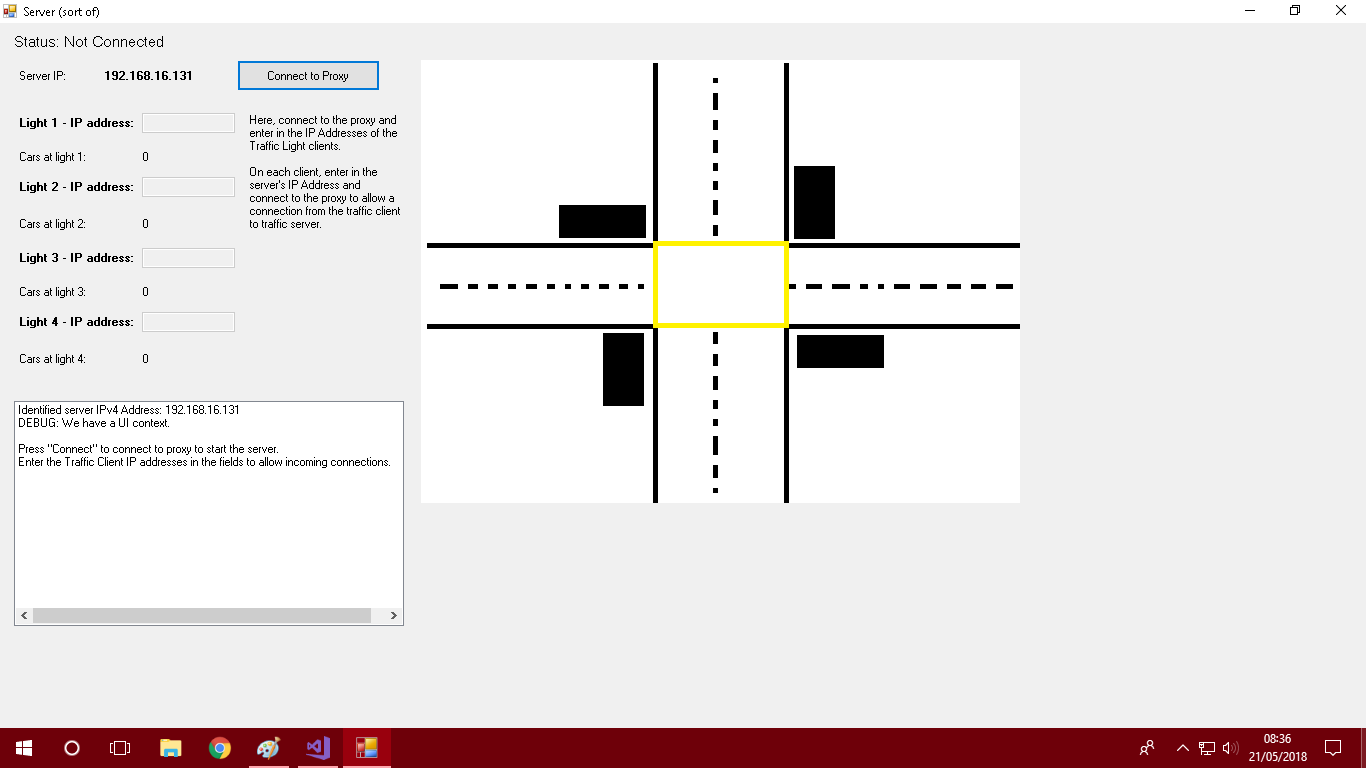
To run the application the client can send any number of car one by one (like in a real world) and to confirm a car has been sent and received by the server messages are displayed both in the server and client side.  For a better understanding of the situation an animation was created for the user which really stands out as the animation was fully raw coded without the use of any DLL or vector which was tough for animating in a real time server.

The server has a time ticker which reads picks up the number of car on each road thus decides to which road should be allowed depending on which road hits the number of 10 cars first, light message which is also sent as string message are being displayed on both side - client and server side.

After cars being allowed to leave the client side is reset to 0 so that the user can send in more car.

The light system displays Red until ten cars are on the que and a timer was set to check the number of cars on each road deciding which road’s car should be allowed to go or halt. Each IP connection is being taken as one road of a cross section, making the program smoother to run on server side.

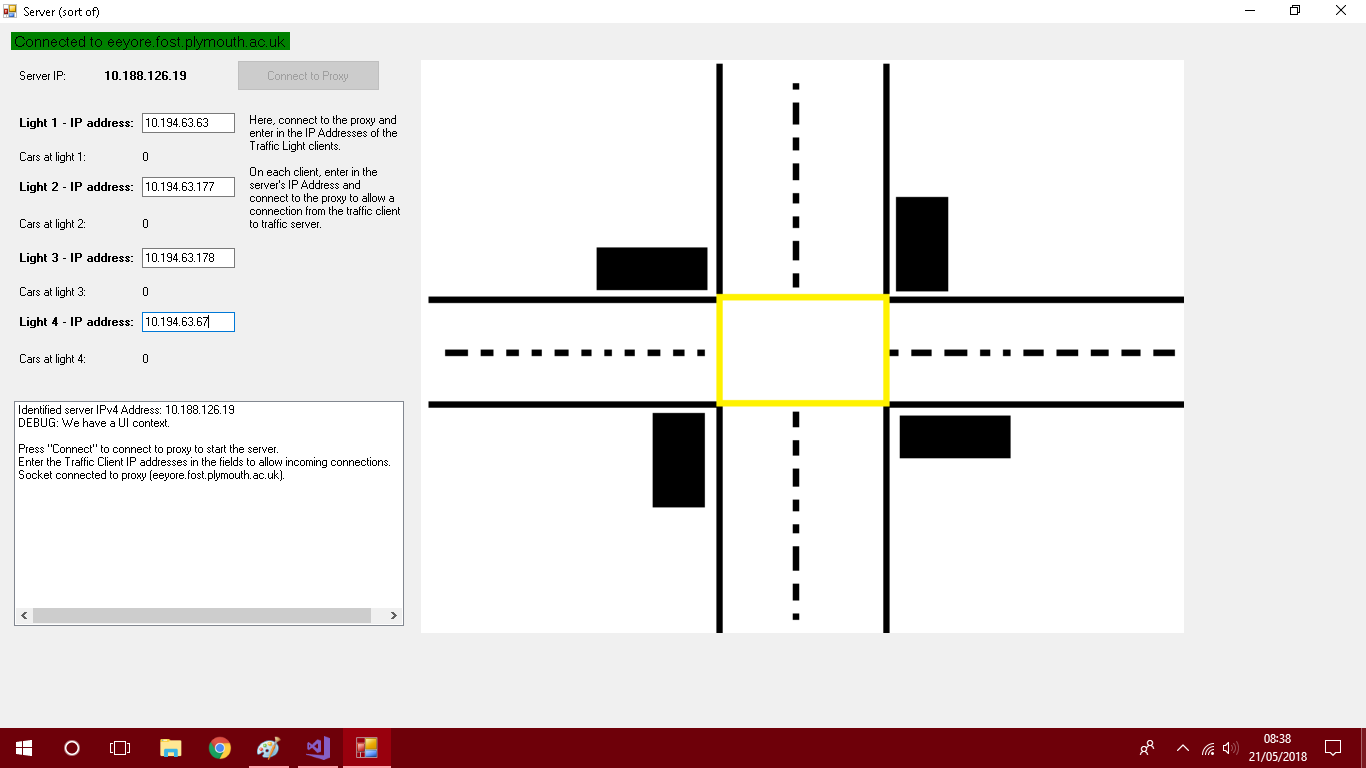
# Usage of Server & Client



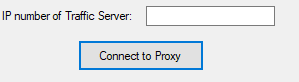
**Traffic Light Server** -

*The window you are presented with upon opening the server application.*

**1)** Press the “Connect to Proxy” button to connect to intermediate proxy that will communicate between clients and this server.

**2)** Enter in the IP addresses of the clients (the 4 client programs either is open on the same computer as the server or on different computers with different IP addresses) to allow incoming connections, as shown below.

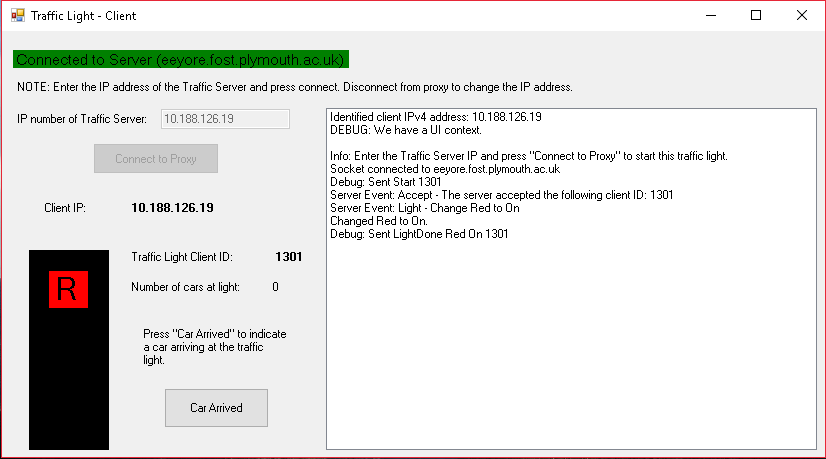
**3)** On each client application, enter the IP address of the server (as stated on the server application) in the textbox given alongside the “IP number of Traffic Server” prompt (see image below). Press “Connect to Proxy” to connect to intermediate proxy which will send connection request to the server alerting of the client’s presence.



**Traffic Client:**

Enter the server IP address textbox and press connect to proxy.

**4)** Upon a successful connection with the server, the server will return a client ID for the Traffic client to communicate with and an initial red traffic light will display. The following window below illustrates a connected client.

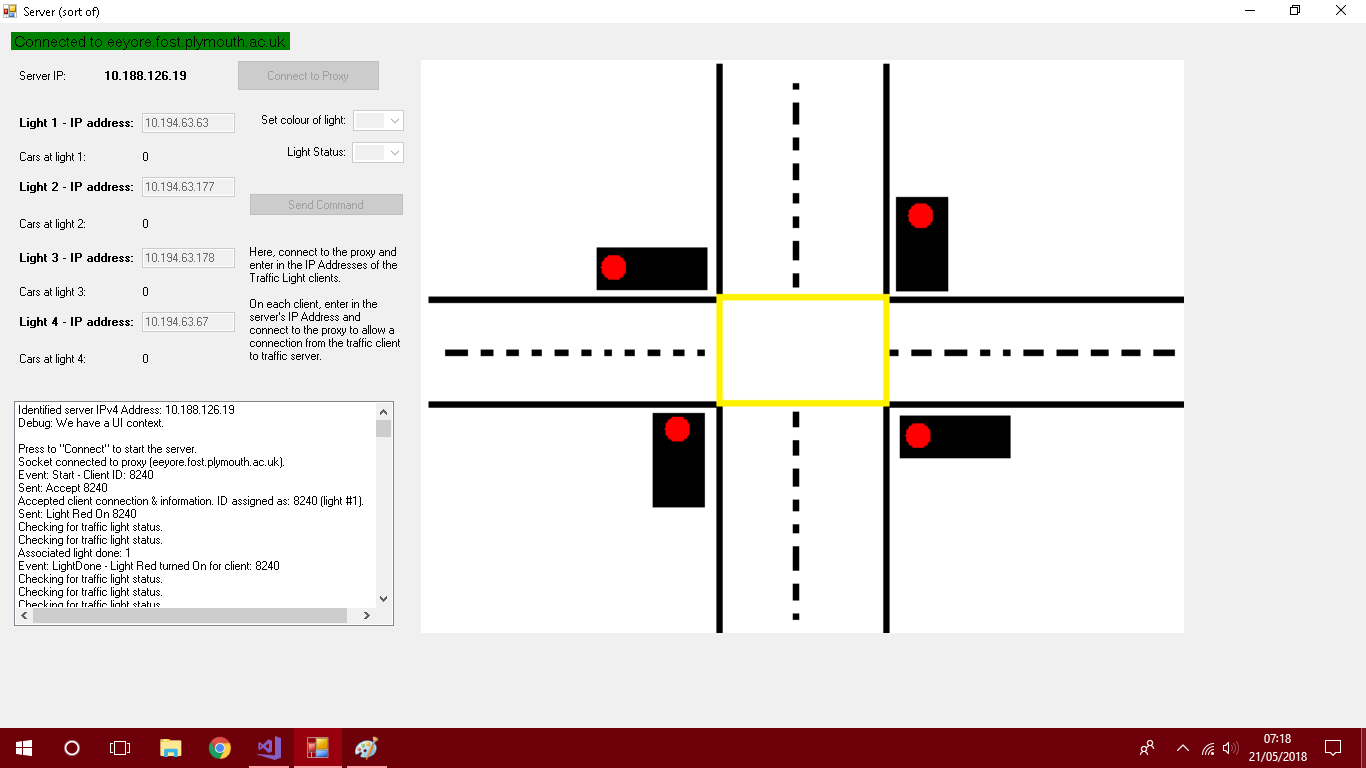


**Traffic Client:**

The client has a connection to the server via the proxy and the server has assigned

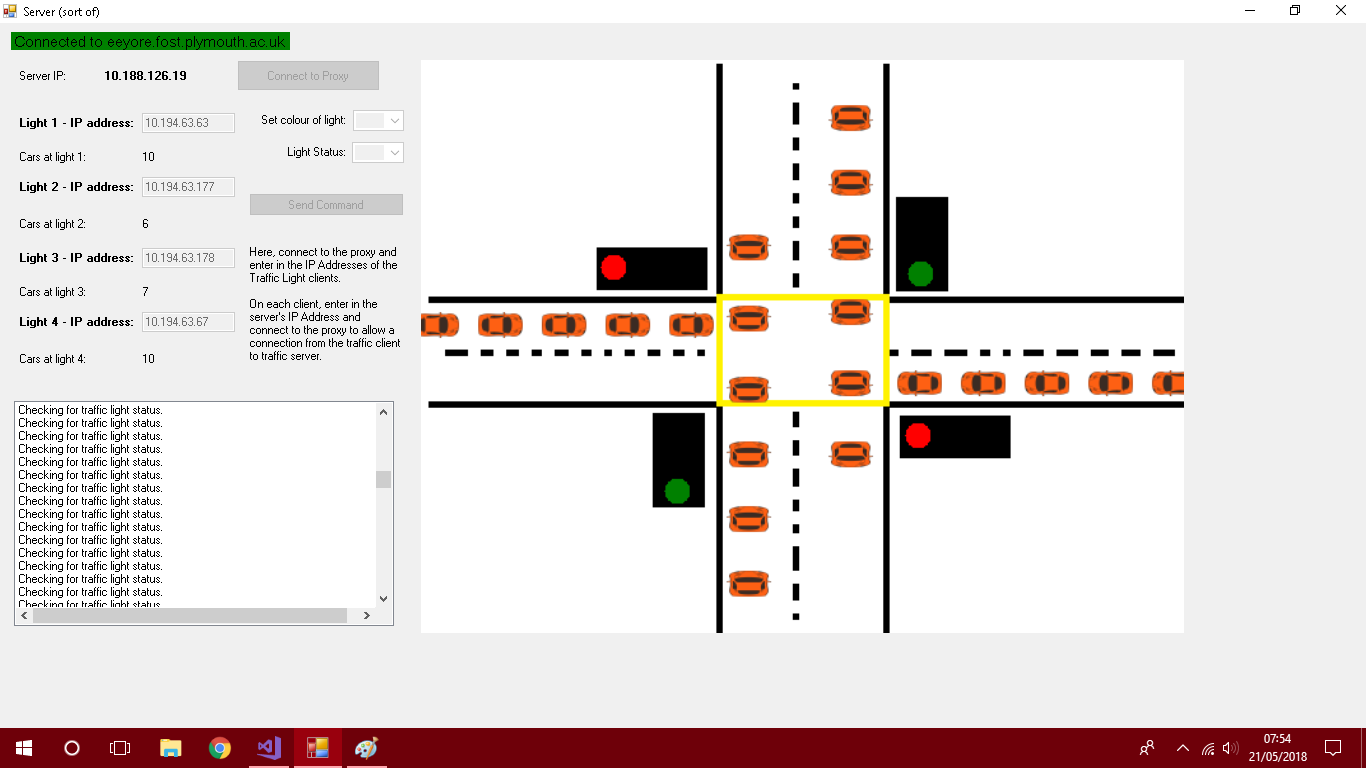
it a *client ID* of 1301 to distinguish it from any other client.

**5)** Once all 4 clients are connected in this manner, the server will display all the red lights to indicate that all 4 traffic clients are ready and recognised by the server (image below).



**6)** The **“car arrived” button** on each client allows for the simulation to begin, i.e. pressing the car arrived button 10 times indicates the arrival of 10 cars at the traffic light which the client represents on the server.

As shown below, this will cause a light change sequence and thus the traffic will flow.



**Traffic Server:**

On the server we simulate what pressing the car arrived on various clients will look like; cars will arrive and stop at the traffic light if it is red, if green they will proceed on the same road.

# Application Layer Protocol

When a connection is made between the client and server, string messages are used and transmitted in packets to communicate.

## Client

The client handled a variety of calls from the server for several actions i.e. connecting and confirming a client ID to communicate with or initiating a change of light sequence which would allow cars to go for the traffic light the client was controlling.

The server’s call and what the client does as an action is defined below.

|  |  |  |
| --- | --- | --- |
| **Server Call** | **Format** | **Purpose** |
| “Accept” | [IP Address, “Accept”, Client ID] | A confirmation of connection message from the server indicating that the client is recognised and can further communication. |
| “Reject” | [IP Address, “Reject”, Client ID] | A rejection of communication from the server; this is often sent when the client is not one that is permitted to make a connection to the server i.e. when the IP address of the client is not listed in the allowed connection or if there are already more than 4 clients connected to the crossroad. |
| “Light” | [IP Address, “Light”, Colour, State, Client ID] | Indicates a traffic light needs to be turned on or off. The colour is specified within the packet along with on or off to indicate the desired state. |
| “LightSequence” | [IP Address, “LightSequence”, Client ID] | A request to the client to begin the sequence to go from red to green and back to red. |
| “CarUpdate” | [IP Address, “CarUpdate”, Count, Client ID] | Allows the server to send an updated count of the number of cars that are present at the traffic light at any given time. |

## Server

Similarly, the server has a specific set of calls that it will accept, as defined below these allow for each client to communicate information to the server.

|  |  |  |
| --- | --- | --- |
| **Client Call** | **Format** | **Purpose** |
| “Start” | [IP Address, “Start”, Client ID] | Allows a client to request the start of a conversation with the server and the client provides the server with a client ID for it to talk back to it with. |
| “Car” | [IP Address, “Car”, Count, Client ID] | Alerts the server, from the client, of the latest count of cars at the traffic light the client oversees. |
| “LightDone” | [IP Address, “LightDone”, Colour, State, Client ID] | Notifies the server that a light has been turned on in the client. Contains the colour of what the light is and its current state so that the server can also reflect these changes on its graphic. |
| “LightSequenceDone” | [IP Address, “LightSequenceDone”, Client ID] | States to the server that the light sequence that the client had been ordered to perform has been completed. |

# Evaluation

Overall the application is a simulation of real world traffic lights, we are satisfied with our application as the server manages to work in co-ordination with the client and conveys the information accordingly via a working animation thus helping the user to understand the actual situation of the crossroads.

If we could make the program more efficient it would a perfect server application, but still as our server if capable of handling multiple clients on various IP address and react accordingly, we believe it is quite a robust solution.

More over the application really stands out because with every command executed, both the server and the client display an output of the specific event that has occurred on the client/server making it easier to use. Moreover, the server updates in waves which helps user to know how many cars had been sent in each wave.

Finally, there were some issues that the application faced during development. Due to the communication of both server and client, there were times at which application appeared slow i.e. the servers count of the current cars on each road may be delayed due to the processing of the messages. In addition, one issues were that of animations. Garbage collection with bitmaps was essential without it, we faced an issue of excessive memory usage when using images rendered to the panel.

## Further Development

If there was a scope to develop the application further we would have created separate threads for each connection without blocking the main thread (which is, for any application, not good).

Finally, we would have changed the simulation into more realistic situation where emergencies like ambulance or creating a 2-way road rather than a one-way road. Another possible addition to the application would be that of a general timer which would not make the lights wait for until 10 cars.

Another potential expansion to the program was that of multiple crossroads handled by one server using a dropdown box of crossroad id’s and with the clients connecting with a crossroad id as well. This would lead onto the fact that each crossroad would be an object with its own connection threads thus a server would be able to handle many crossroads at once.

# Contributions

The development of the solution relied on using pair programming as well individual work which was discussed before-hand and then carried out. To get the server working with the client *Mr. Tajwar* created graphics used in animations on the server and tested this functionality and the logic behind drawing the animations on the server successfully.

Contributions to the application layer protocol was made by *Mr. Biju* and the work of server and client was split up regularly when testing and make modification to the protocol where *Mr. Tajwar* would be programming for the client and *Mr. Biju* would work on the server. This ensures that one person could be generating a said call from the server to client and the programmer at the client application would be handling the event that was generated by the call. By programming in this manner work was dealt with appropriately and the set targets for each programming session would be met.

When working alongside each other, problems that were encountered were discussed and often researched together, and separately, to find a path to overcome an issue. Code was also peer-reviewed to ensure that it would be appropriate.

To finalise, both contributors worked to prioritise the main tasks and decomposed each problem step-by-step to get to the next task. The overall product was a solution that was robust due to help that each contributor provided for each other in the project.