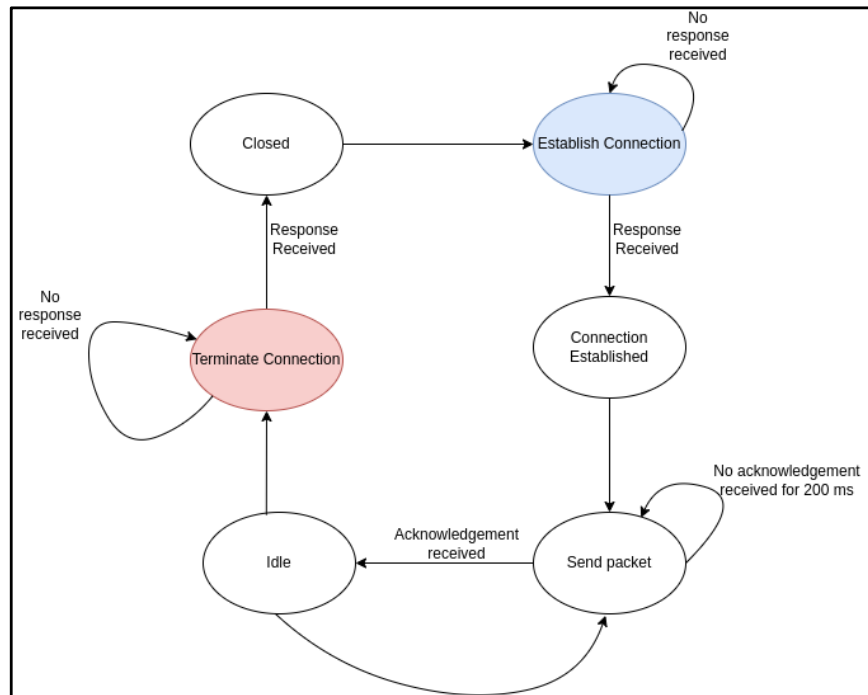


## Design

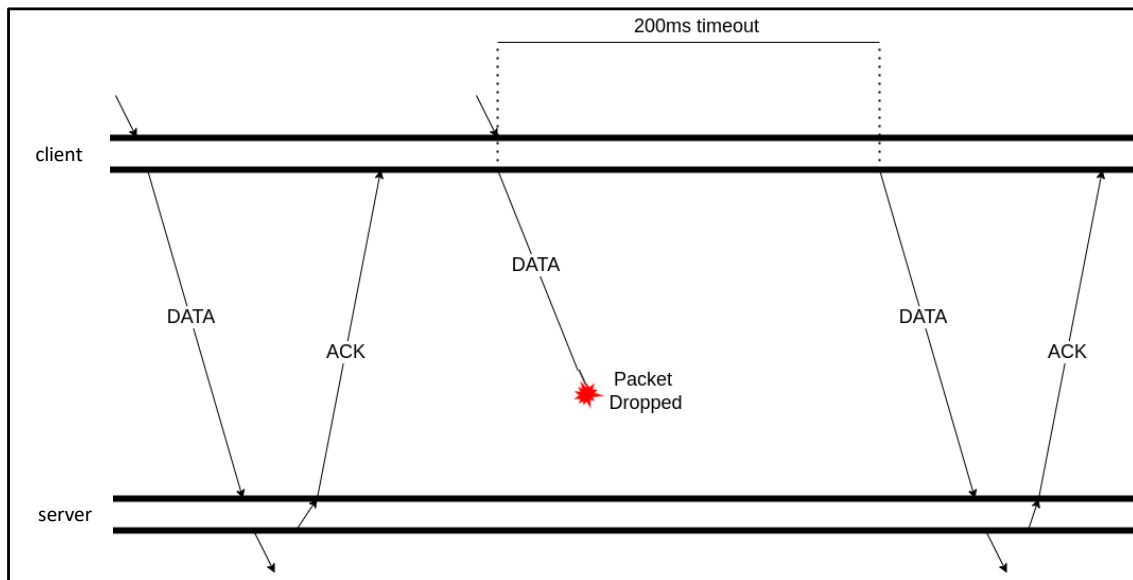
### FSM for connection management



*Figure 1: FSM Diagram*

Figure 1 Shows the FSM designed for the connection management. There is a simple handshake for establishing a connection (shown in blue) where it will continue to try and establish until a response is received. It will only move on to a packet transferring state once it receives a response.

Similarly there is a state with a handshake for terminating the connection (shown in red) that will only close the connection till the response is received.

**Data transfer using an idle-RQ mechanism.**

*Figure 2: idle-RQ data transfer diagram*

Figure 2 shows a diagram for the data transfer between my client and server. The client transmits data in the DATA packet and upon successful transfer the server sends acknowledgement packet ACK back. If the client does not receive acknowledgement within the fixed 200ms time-out it will re-transmit the DATA packet.

## API Protocol

### Packet Codes

For my protocol, specific numbers when stored in the binary bytes for the packet act as flags to the server or client on what type of packet it is. And they are as follows.

ESTABLISHCODE: 1

TERMINATECODE: 2

ACKNOWLEDGECODE: 3

### Server

The server's idle function contains a while loop that will end when the connection is terminated. The server listens for packets and when it receives one it checks the bytes.

A value of 1 means the client is trying to establish a connection and so the server sends a packet with a 1 back to agree.

A value of 2 means the client is trying to terminate the connection and so the server sends a packet with 2 back to agree.

Any other value than these two and the server knows that the packet sent is a DATA packet, and so the server sends an ACK packet back which has a value of 3.

### Client

The client contains three notable functions:

*establishHandShake, sendData, terminateHandsShake*

Which all call the *idleRQ* function which sends a packet with the relevant payload and then checks the received packet for a specified code. This function will also retransmit packets if a set timeout time is reached without receiving an acknowledgment.

## Testing

### Important Note

The test programs contain a constant MY\_PORT, which would need to be changed to whoever is running the program's personal port number.

### Simple Test

I have two test programs, testClient and testServer. They both test and provide output on the connection establishing, data transferring and connection terminating of the client and server api's.

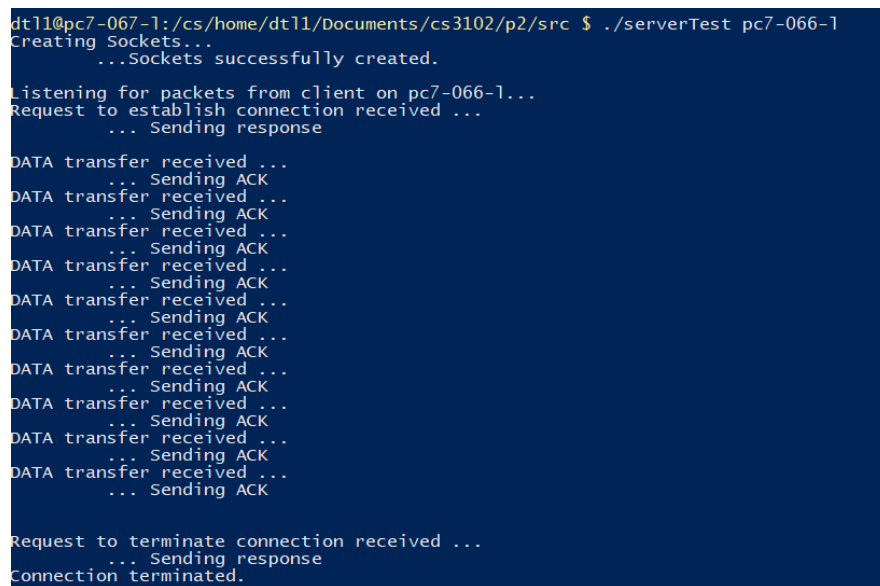
testClient makes a connection to the server and sends 10 example packets of data before terminating the connection.

Each program takes one command line argument, the FQDN of the machine the opposite program is running on.

For example, to test the protocol you would need two sessions running on two different machines. First, on either machine, run "make test" to compile the test programs.

Now, on the first machine run ./testServer <name of second machine> and on the second machine run ./testClient <name of first machine>.

An example of this is shown below:



```
dt11@pc7-067-1:/cs/home/dt11/Documents/cs3102/p2/src $ ./serverTest pc7-066-1
Creating Sockets...
...Sockets successfully created.

Listening for packets from client on pc7-066-1...
Request to establish connection received ...
... Sending response

DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
DATA transfer received ...
... Sending ACK
Request to terminate connection received ...
... Sending response
Connection terminated.
```

*Figure 3: Output of the test program for the server api*

```

dt11@pc7-066-1:/cs/home/dt11/Documents/cs3102/p2/src $ ./clientTest pc7-067-1
Creating Sockets...
...Sockets successfully created.

Attempting to establish connection with the server on pc7-067-1...
...Server connected in less than 1ms
Connection established.

Sending some data to the server...
...Server acknowledged the data in 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending request to terminate connection...
...Server agreed in less than 1ms
Connection terminated.

```

*Figure 4: Output of the test program for the client api*

If I instead run the client before the server, you can see an example of the handshake working to establish the connection.

```

dt11@pc7-066-1:/cs/home/dt11/Documents/cs3102/p2/src $ ./clientTest pc7-067-1
Creating Sockets...
...Sockets successfully created.

Attempting to establish connection with the server on pc7-067-1...
...server didnt respond within 200ms, re-attempting handshake...
...server didnt respond within 200ms, re-attempting handshake...
...server didnt respond within 200ms, re-attempting handshake...
...server didnt respond within 200ms, re-attempting handshake...
...server didnt respond within 200ms, re-attempting handshake...
...Server connected in less than 1ms
Connection established.

Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending request to terminate connection...
...Server agreed in less than 1ms
Connection terminated.

```

*Figure 5: Output of the test program for the client api showing the handshake at the begging to establish the connection.*

### Simulated Packet Loss Test

A more advanced test involves using a third machine and slurpe-3.

On machine 1, run `./clientTest <name of machine 2>`

On machine 2, run `./slurpe ./slurpe-3 loss <name of machine 1> <name of machine 3>`

On machine 3, run `./serverTest <name of machine 2>`

This will route packets through slurpe-3 and introduce an element of loss, to test the idle-RQ re-transmission feature.

An example of this is below:

```
dt11@pc7-066-1:/cs/home/dt11/Documents/cs3102/p2/src $ ./clientTest pc7-062-1
Creating Sockets...
...Sockets successfully created.

Attempting to establish connection with the server on pc7-062-1...
...Server connected in 67ms
Connection established.

Sending some data to the server...
...200ms timeout reached, re-transmitting data...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...200ms timeout reached, re-transmitting data...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...200ms timeout reached, re-transmitting data...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...200ms timeout reached, re-transmitting data...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...200ms timeout reached, re-transmitting data...
...Server acknowledged the data in less than 1ms
Sending some data to the server...
...Server acknowledged the data in less than 1ms

Sending request to terminate connection...
...Server agreed in 1ms
Connection terminated.
```

*Figure 6: Output of the test program for the client api.*

As you can see in figure 6, of the 10 packets of data sent, it would appear the first 2 and the last 3 reached their timeout and had to be re-sent.

```
dt1@pc7-062:1:/cs/home/dt1/Documents/cs3102/p2/src $ ./slurpe-3 loss pc7-066-1 pc7-067-1
** slurpe-3 loss pc7-066-1 (138.251.29.76) pc7-067-1 (138.251.29.77) - port=21929, max_msg_size=1400 bytes, max_q_size=128 packets

-- packet dropped

-- packet dropped


-- packet dropped

-- packet dropped

-- packet dropped
```

Figure 7: Output of slurpe-3 loss, running as a router inbetween the apis.

And as you can see in figure 7, the first 2 packets and the last 3 were dropped, this coincides with the behaviour in figure 6.

```
...Server acknowledged the data in less than 1ms
Sending request to terminate connection...
...server didnt respond within 200ms, re-attempting handshake...
...server didnt respond within 200ms, re-attempting handshake...
...server didnt respond within 200ms, re-attempting handshake...
...Server agreed in less than 1ms
Connection terminated.
```

Figure 8: Output of the test program for the client api.

Upon another run of this same test, figure 8 shows the termination handshake working as the first 3 attempts to terminate were lost by slurpe-3.

## Critique

As I am effectively reserving binary packet values 1,2 and 3 as codes, the protocol would act unexpectedly if it tried to send a 1,2 or 3 in a DATA packet. To avoid this if the protocol were fleshed out and used in a scenario where this could happen, then I could implement some system to let the server know that the client is trying to send data and not an establishment or termination request. One way of doing this could be using the last 3 possible numbers of the byte value for the packet payload to be 1 2 and 3 instead.