

# CS380 — Project 6

February 15, 2016

Due: Wednesday, February 24, 2016 (60 points)

## Description

In this project, we'll be (mostly) implementing TCP's connection establishment and connection teardown procedures. It will be similar to the previous project in that you must implement TCP on top of IPv4. My server will act as the passive participant and wait for you to initiate a connection.

After the connection has been established, you will send 12 packets in the usual fashion (starting with 2 bytes of data, then doubling each time). We won't implement the sliding window algorithm, instead I will simply reply as I did in the previous project by sending response codes.

After you receive the response code for the final packet, you must then initiate the connection teardown. Once the connection has been successfully closed, your program should exit.

For the connection establishment and connection teardown, I will respond with TCP headers, but not IPv4 headers or TCP data. Therefore, your program must be able to parse a TCP header but not the rest of the packet.

The window size and urgent pointer fields should be all zeros for every packet. You can choose any source and destination ports that you wish.

The process is as follows:

1. You will send a complete IPv4 packet containing a TCP header that has a randomized sequence number and the SYN flag set.
2. I will first respond with a 4 byte code to confirm the correctness of your SYN packet, if it's 0xCAFEBAE then we'll continue.
3. I will then respond with a TCP header with the SYN and ACK flags set and a randomized sequence number.
4. You will send a complete IPv4 packet containing a TCP header that has the value of your sequence number + 1, the ACK flag set, and an acknowledgment of my sequence number + 1.
5. I'll respond with another 4 byte code to confirm the correctness of your ACK, if it's 0xCAFEBAE we'll continue.
6. Send all 12 data packets with proper sequence numbers. I will respond with 0xCAFEBAE for each one if they are correct. Remember that TCP gives each byte of data its own sequence number! You should have started the data sequence number by incrementing the value you used in your ACK packet.
7. Begin connection teardown by sending a packet with the FIN flag set.
8. I'll respond with a 4 byte code confirming the correctness of your FIN packet.

9. I will respond with a TCP header with the ACK flag set to acknowledge closing the first half of the connection.
10. I will then send a TCP header with the FIN flag set to begin closing the second half of the connection.
11. You will respond with the ACK flag set to confirm closing the second half of the connection.
12. Finally, I'll respond with a final 4 byte code confirming the correctness of your ACK.

Code	Reason
0xCAFEBAFE	Correct data packet
0xBAADF00D	Problem with IPv4 portion of packet (see project 3 to fix!)
0xCAFE00D	Unexpected flags set in TCP packet
0xDEADC0DE	Invalid TCP checksum
0xBBADBEEF	Incorrect TCP data length
0xFEEDFACE	Incorrect sequence number
0xFEE1DEAD	Bad data offset
0xBAAAAAAD	Incorrect acknowledgment in three-way handshake

The server is running on `cs380.codebank.xyz` on port 38006.

Your IPv4 headers will have the same restrictions as project 3. Keep in mind that the TCP checksum is calculated using a similar method to project 5. It includes the TCP data and a “pseudoheader” including part of the IPv4 header.

## Submission

### Submission

1. On [codebank.xyz](https://codebank.xyz), create a project as your user named CS380-P6. Follow this naming convention precisely including case.
2. Make sure you add the reference to the remote repository in your local project with:

```
$ git remote add origin https://codebank.xyz/username/CS380-P6.git
```
3. Your project should have a main class named `TcpClient` in a file named `TcpClient.java`. You can have other files or classes, but it should successfully compile and run by simply using:

```
$ javac TcpClient.java
$ java TcpClient
```