Mandatory hand-in 2

TCP/IP Simulator in Go

a) What are packages in your implementation? What data structure do you use to transmit data and meta-data?

We assume that "package" was supposed to be packet. In our implementation, we have made a struct implementation of TCP packets. Our struct is defined as follows:

```
type message struct {
      flag    string
      seq     int
      ack     int
      id      int
      data    string
}
```

The 'flag' is a string type, that represents the flag of each packet. When the client initially sends a synchronization request, it has the value: "syn", and as the protocol progresses it changes into "syn ack", “ack" and finally “FIN”.
The 'seq' is an int type, that represents the sequence number.
The 'ack' is an int type, that represents the acknowledgment number.
The ‘id’ is an int type. We use it as a counter. More on this follows in section c and d.
These 4 values together make out the meta-data. The ‘data’ is a string type, that holds the data itself. This, together with the meta-data is the full packet.

b) Does your implementation use threads or processes? Why is it not realistic to use threads?

The full implementation, is itself a process, however our implementation utilizes threads to solve the problem. Using threads to simulate the Transmission Control Protocol (TCP), is not realistic, for several reasons. Fundamentally, TCP is meant for transmissions between machines, whereas threads run on the same machine, and even share some resources, such as a stack. On top of sharing resources, running on the same machine, also removes obstacles such as network latency and reliability. Threads are not exposed to the same attacks on security in the same way either, and are not expected to be able to scale to extreme numbers of connections.

c) In case the network changes the order in which messages are delivered, how would you handle message re-ordering?

To handle message re-ordering, we have added a counter to the data. The counter works by incrementing each time the client sends a packet containing data to the server. Using this, the server can re-organize the received data, to follow the order of the counter.

d) In case messages can be delayed or lost, how does your implementation handle message loss?

To handle message loss, we again use the counter. Using the counter, the server can tell that a message was lost, if the pattern of the counter is broken. We have not implemented a solution to message loss, but lets the user know that a problem occurred. When sending the final packet with the “Fin” flag, we send the counter in the data-field, to make sure the server knows the full pattern.

e) Why is the 3-way handshake important?

The 3-way handshake is important to ensure there is a strong connection between client and server. As opposed to the Internet Protocol, TCP ensures that there is a connection between client and server, before it starts sending data. This makes the connection stronger. In addition, the sequence and acknowledgement numbers are used to identify the connection, so multiple data packets can be sent, and the server will know they are related.