Byron Garcia & Marcus Loe

CSE 160: Computer Networks

12/3/18

Discussion Questions

1. The chat client and server application as described above uses a single transport connection in each direction per client. A different design would use a transport connection per command and reply. Describe the pros and cons of these two designs.

A single transport connection would only allow one command to be sent along the line at a time. Therefore, it would be much slower but the message would not get cut off. With a different connection for each, it would be much faster, but less reliable because SYN/ACK might get lost and the packets will either be lost or not come in order.

2. Describe which features of your transport protocol are a good fit to the chat client and server application, and which are not. Are the features that are not a good fit simply unnecessary, or are they problematic, and why? If problematic, how can we best deal with them?

The transport protocol only work with strings, it would be better if we could send objects over the transport protocol so we could send more data points to the protocol. We could write a function that parses an object to a string and then back into an object.

3. Read through the HTTP protocol specification covered in class. Describe which features of your transport protocol are a good fit to the web server application, and which are not. Are the features that are not a good fit simply unnecessary, or are they problematic, and why? If problematic, how can we best deal with them?

Our transport protocol is very simple, and it needs more work to actually become a web server application. However, the core features are implemented and will send packets to different nodes.

4. Describe one way in which you would like to improve your design.

So far we've been able to modularize the program, so features can sort of easily be implemented and of course be removed. However, we'd like to make it more understandable to other people, as when we present our projects we needed to go back through and re-comment what we implemented.

Write up

TCP

In order to discover the sockets and use them as part of the project, we had to start with the connection. Once a client connects to a server, we copy the listening socket and create a new one for the connection to take place and add it to the list of sockets on the client and server side. Sockets are now identified when we just find a socket with corresponding source and destination port values. We don't really use file identifiers because we're pretty lazy.

Data transmissions were changed a bit. Since we are send numbers that are already ordered, it doesn't matter. Therefore, buffering the data is easy because everything corresponds to each other. Also, since we never actually remove from the client/receiving end's buffer, our sliding window only checks if the buffer has enough space. Since we haven't reached the max size, therefore we haven't reached the max payload size. Each window has a set amount of data it can take. Since we check before we send, we should be okay. However, since nesC is different, we can't have dynamic timers for each and every packet. Instead, we just sent an array of data to be sent to the client/server that each window can take. Since we can't differentiate which packet is with which timer that nesC cannot do, we just resent the packet if its lost. In addition, sending multiple packets at a time is harder without the timer. Since we use a sequence number to keep track of which ones are lost and which ones make it, we just resend the lost packets instead of trying to backtrack.

For connection teardown, we just did a simple three way handshake, but in reverse. The big downside of such is that we might lose a packet here and there, but that should be okay since 99% of it make it. We might also have offset to help with this. In addition, the FIN signal shouldn't really cut off since the packet already made it. As told previously, building project 3 properly in order for future projects is key. In addition, we realized that nesC doesn't have the same functions as C, so we were limited by the scope of what we could do. So far we've modularized it to the point in which it somewhat works seamlessly, and removing/adding things is much easier.