CSE 160 Computer Networks

Project 4

Discussion Questions & Report

By: Abdias Tellez & Renato Millan

Design Decisions

One challenging aspect of implementing a client/server model for this project was the bidirectional communication. Our TCP implementation was unidirectional; therefore, our solution was that each server and client would need 2 TCP connections between each other in to communicate.

Foremost, the test python scrip was almost identical the test scrips for project 3. Each server would need to accept connections like in project 3. When the connection is established, we needed 2 pointers (receive and send) for each buffer to keep track of the of the last data read and written. For each bidirectional connection to clients, we needed to store the file descriptors for the listening socket and two.

In addition, a server on any node is run on boot, which allows clients to connect to it immediately. The chat commands(hello,msg,whisper,listusr) is passed and processed by the chat application module and string comparisons are made. We made fail case so it can indication the user that the command was misspell.

The chat application is modular, and it interacts with the transport layer as an interface. The TCP is abstracted away from the chat app because it is the backend for the chat application; it only connects, reads, and writes. This was done to simulate a real-world program developed on a Unix-like system, like previously. The biggest difference was TinyOS's asynchronous nature and our unidirectional TCP implementation, which added to the application's complexity.

Question

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.

For single transport connection:

Pros: More reliable, as this would only allow one command to be sent along the line at a time.

Cons: Much slower as compared to the other design.

For transport connection per command reply:

Pros: Much faster, as compared to the other design.

Cons Less reliable, packets could be lost or placed in different orders. SYN/ACK could also be lost.

2)

Describe which features of your transport protocol are a good fit to the chat client an 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 features include the ability to send over strings. This could be problematic, in the case that we wanted to send other packages that include data structures outside of this datatype, such as objects. The best way to deal with this would be to implement a function that can convert from string to object and vice versa.

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?

For sending and receiving data, our transport protocol leverages dependability and has varied purposes and priorities. Our implementation of project 4 does not meet the requirements to be a web server application. The essential functionalities have been implemented, and packets will be sent to various nodes.

4)

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

In future iterations of this project, I would improve the chat limitations. We're currently limited by the size of the string in the chat. The message size is fixed through the command handler, and we would like to change that, so our messages can be full on paragraphs. We would also like to include the documentation and comments of our code.