COM SCI 118 SPRING 2018
Computer Network Fundamentals

Project 2: Simple Window-Based Reliable Data Transfer

Names: Nicholas Rivera, Dennis Zang
UIDs: 404465866, 704766877

Client/Server Design Overview

To implement a TCP-like file-transfer protocol over UDP, we had to encapsulate the TCP header and data into UDP packets. Because UDP is connectionless and there is no way to delineate special data in UDP packets, we had to come up with our own protocol so that the client and server could understand the data that they send each other.

Difficulties and Implementation

Creating the Connection:

We create the connection by creating UDP sockets. Given a port number, we bind a UDP socket to that port number. The client does not have a specific port number, so the OS chooses one for us. The client also converts the IP address of the server into a form understandable by the OS.

3-way Handshake:

We need to initiate the 3-way handshake to ensure that both parties are aware of the connection. The handshake is initiated by the client, which, repeatedly in a loop on timeout, sends a packet with the SYN flag set to the server, which is listening for connections. The sequence number is also set to a random number on a scale from 1 to (`MAXIMUM_SEQUENCE_NUMBER` - 1), and the acknowledgement number is 0 (since the client is not acknowledging anything). When the server is aware of the connection, it responds with a packet with the SYN and ACK flags set, with the sequence number set to a random number in the same range, and with an acknowledgement number equal to the next expected packet (which is the sequence number of the client, plus one). Finally, to complete the connection, the client responds with the last packet in the three way handshake, with the ACK flag set and the acknowledgement number being the increment of the server's sequence number. We piggyback this information when we send the client's file request to the server.

Reading and Sending the File:

The client sends the file name to the server. The client tells the server that it is done sending the file name by then sending a FIN packet. The client tells the server, "I'm done sending data and will now wait to receive my data." After getting the filename from the client, the server reads the file all at once and places the bytes of the file in a vector. The server then takes a chunk of bytes from this vector, wraps them up in a packet, starts a timer, sends the packet, and then places the packet in a list of packets that are currently in transit. If the list of packets in the queue gets too large (as dictated by the client's receive window), the server will hold off on sending. If the timer goes off on the earliest sent packet in the queue, the server will resend that packet.

Exchanging Packet Sequence Numbers and Acknowledgements:
The sequence numbers tell the client the order of the packets that the server is sending. If the packets are reordered, the client will not buffer the unordered packets. The client will drop an out of order packet. If the server re-sends a packet that the client has already seen, the client will resend an acknowledgement for that packet. Acknowledgment numbers tell the server which packets the client has seen. In our TCP implementation, acknowledgement numbers are not cumulative: The server expects one ACK message for each packet. This makes sure that when sequence numbers wrap around, all acknowledgements are handled correctly.

Closing the Connection:
To close the connection, the server program checks if all of the data has been sent. This is done by keeping track of how much data is left in a list of bytes that we empty out as we send. When the list of bytes is empty, the server sends its FIN message on the last packet with file data. Two things then happen. First, the server keeps track of how many re-ACKs it sends after it has already sent the FIN message. This took care of our biggest difficulty: Sometimes, the client will receive the FIN message, send its ACK, then shut down before it is able to resend that ACK. The server will shutdown after it has unsuccessfully tried to re-send its FIN message. When the client sees the FIN message, it will wait twice the timeout interval. This makes sure that the server has seen the client's ACK message for the FIN packet. Our TCP implementation is different from the normal TCP shutdown because the FIN packet is piggybacked on the sending of the the last data packet.

How to Build and Run

Requirements:

- A copy of g++8

To build the program, just type `make`.
To run the server, give as the first argument the port number the server should use. For example, you can run the server like this:

```
./server 3874
```

To run the client, give as the first argument the IP address of the server. The second argument is the port number of the server. The third argument is the name of the file that you want the server to send to you. For example, you can run the client like this:

```
./client localhost 3874 hello.txt
```

TCP Header Format

To implement a TCP-like file-transfer protocol over UDP, we had to create special headers for our packets. The headers had the format:

Sequence Number
Acknowledgement Number
Receive Window
Length of Data
ACK Flag
SYN Flag
FIN Flag
Data

The `Length of Data` field tells the receiver how much data to read out of the `Data` field. The `ACK Flag` tells the receiver if the `Acknowledgement Number` field has an acknowledgement number in it. The `SYN Flag` field is true if the sender is setting up the connection. The SYN flag tells the receiver that the Sequence Number that the sender is sending is the initial sequence number. The `FIN Flag` field is true if the sender has no more data to send. The `Acknowledgement Number` tells the receiver which byte of data the sender is waiting for next. The `Sequence Numb`er tells us the number of the first byte of data in the `Data` field.

References:

https://stackoverflow.com/questions/13445688/how-to-generate-a-random-number-in-c
https://stackoverflow.com/questions/25520729/most-efficient-way-to-split-a-vector-into-several
https://stackoverflow.com/questions/5328070/how-to-convert-string-to-ip-address-and-vice-vers
a
https://stackoverflow.com/questions/5773088/making-a-timer-in-c
https://www.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html