Members

Ryan Mercer 803817943
Spencer Brett

Implementation

Packets

Description	Number of bits	Data Type
Sequence Number	32	uint32_t
Ack Number	32	uint32_t
ACK	1	1st ms bit of uint16_t
Last	1	2nd ms bit of uint16_t
Close	1	3rd ms bit of uint16_t
(unused)	3	4th-6th MSB of uint16_t
Size	10	LS 10 bits of uint16_t
Checksum	16	uint16_t

12 bytes of header, 1012 bytes of body
We assumed that the UDP would keep track of the port number and ip address, so we did not include these in our header. If this was going to be a full TCP implementation, we would have included 4 additional 4-byte-words for the port number and ip of the sender and receiver.
ACK: identifies a packet as an acknowledgement
Last: identifies the last packet in a packet set, allows readTCP to exit
Close: identifies packets specifically for closing a connection
Size: the number of data bytes in the packet

Timeout

We used the SO_RCVTIMEO option when setting up our socket with a timeout of 100 ms. We estimated that a fast RTT = 20ms, so 5x should be adequate. With this timeout method, we simply resend all unacknowledged packets in the current cwnd when readfrom returns -1.

Protocol

We implemented GoBackN which uses a single timer that keeps track of the oldest packet still unacknowledged. The number of packets sent in parallel is equal to cwnd. The sending function fills up all available cwnd spots with file packets then waits for all of the packets to be ack'd before adding the next batch of file packets to the cwnd.
One difference from the typical protocol is that we did not handshake at the beginning, but simply had the client used the writeTCP function and server use the readTCP function to make sure the file request was sent with large error probability.

Difficulties

timeout

we were not sure how to handle timeouts. We came across a select function that took timeout as input. The easiest implementation seemed to be setting the socket setting SO_RCVTIMEO so that readfrom would return -1 when a timeout occurred.

setting bits

We decided to try to be a bit efficient with our packet header by setting individual bits instead of using a whole byte for the ACK LAST, and CLOSE info. This just took some testing and bit masking.

writing large pieces without testing

At times nothing seemed to work and we used a large number of printf messages to trace where issues might be. These were typically 1-off issues where the bounds were not correct or number overflow occurred. This could have been prevented if we tested each function before trying to use it in the larger context.

coding for the end result instead of iteratively

We had the goal of programming iteratively, but it was too tempting to write the code so that it would support future needs. This ended up working out, but it was pretty risky.