CS118: Computer Network Fundamentals
Richard Min (UID: 604451118)
Joanne Park (UID: 104450395)
Spring 2016

Project 2: Simple TCP-like Transport Protocol over UDP

- **Please use our Makefile and Vagrantfile because we use g++5!**
- **Do vagrant provision before use/testing**
- **Base Project executables: targets/server and targets/client_nobuffer**
- **Extra Credit executable: targets/client**

High Level Design of Server and Client

- Both server and client have their own executables. However they only do the argument parsing and socket setup. The majority of the TCP portion of the code is done using TCPManager.cpp. The functions: custom_recv() and custom_send() do the TCP functions for server and client respectively.

- The server (custom_recv() in TCPManager.cpp)
  - First, do SYN-SYNACK handshake with client
  - After the connection is established, initialize a map, indexed by sequence numbers, which buffers the packets that have been sent but not yet ACKed. The window is a portion of this buffer.
  - Begin reading from the specified file and sending packets of it to the client, while adding them to the map buffer. The number of packets sent at a time are determined by congestion control.
  - After having sent a window of packets, wait for their ACKs. For every ACK, increase cwnd. For an ACK, delete its corresponding packet(s), from the window index (the last ACKed packet) up to the received packet. The packet is a retransmission if it is more than the max window size above the window index, otherwise, we slide the window over to the received ack.
  - The server waits 500ms for the ACKs to arrive. If no ACKs arrive, it resends the whole window and restarts the timer.
  - Repeat the previous 3 steps until whole file is sent. Then initiate FIN-FINACK sequence. After having sent the last ACK, the server will wait to make sure the FIN-FINACK sequence is finished.

- The client (custom_send() in TCPManager.cpp)
  - First, initiate SYN-SYNACK handshake with server.
  - Begin receiving data packets from the server until you get a FIN flag. Using the window, if we get the expected sequence number we calculate the next expected sequence number, move the window, and write the received data to disk; otherwise buffer the data (for ./client only)(./client_nobuffer has no buffer and will drop packets). It will send a cumulative ACK back to the server for every packet received.
  - After receiving a FIN packet, do the finish the FIN-FINACK sequence. The client is done and should have received the whole file.

CS118: Computer Network Fundamentals
Richard Min (UID: 604451118)
Joanne Park (UID: 104450395)
Spring 2016

Problems you ran into and how you solved the problems

- HtonS
  - The function recvfrom() seems to convert the byte order for us (depending on if its blocking or not), so converting byte order is done on only for sending packets (using htons())
- Window Indices
- Map manipulation
  - For some reason, the map.erase() function did not work with 2 const iterators, so just used one iterator instead
- Off by one/ Fence post errors
- Overflow + window wrapping
  - Used the TA's slide example, to calculate how to wrap the buffer window correctly.
- Increasing correctly SEQ and ACK numbers across both client and server
- Solved using many print statements to find flaws in logic.

Additional Instructions to build your project

- **Please use our Makefile and Vagrantfile because we use g++5!**
- **Do vagrant provision before use/testing**
- **Base Project executables: targets/server and targets/client_nobuffer**
- **Extra Credit executable: targets/client (uses client buffering)**
- The Makefile builds into /targets the executables: server and client
- /utils contains helper functions

How you tested your code and why

- We ran the files given to us (small.txt and large.txt) with various tc settings. We began with no packet loss at all. Then with packet loss on the client side; then both client and server side. We made sure that the cases worked with the small.txt before moving onto large.txt.

Contribution of each team member

- Richard Min (UID: 604451118)
- Joanne Park (UID: 104450395)
- Richard started the project and laid out the general structure of the code. From there, both of us filled in the code, switching between client and server. Richard also added a buffered client option for extra credit.