We ran the following tests:

- **test_syns**: ran your sender against a receiver that never sends SYNACKs. We expected your sender to send the receiver three SYNs and a RESET.
- **test_fins**: ran your sender against a receiver that never sends ACKs to FINs. We expected your sender to send either one or three FINs, then a RESET.
- **test_normal_small**: ran your sender on a 10KB file against the normal receiver, made sure that the input file and output file matched (using md5sum) at the end
- **test_normal_large**: ran your sender on a 100KB file against the normal receiver, made sure that the input file and output file matched (using md5sum) at the end
- **test_pipeline**: ran your sender against a receiver that never sends ACKs for DATA packets. We expected your sender to pipeline 30 packets (to fill the receiver's 15000B window) and then retransmit those packets forever (although we waited something like 15 seconds). Pipelining 31 packets or 29 packets was still considered correct (i.e. we allowed you to be off by one in either direction to account for can_send_data's vague use of length). We ran on a 17KB file to give you enough data to fill the pipeline.
- **test_reorder**: ran your sender against a modified version of the normal receiver that re-ordered the 10th packet received on a 10KB file. Made sure input and output files matched at the end (using md5sum)
- **test_lossy**: ran your sender with a 10KB file against a modified version of the normal receiver that dropped the 10th packet and 20th ACK received. Made sure input and output files matched.
- **test_smallwindow**: like test_normal_small, but with a 500B window instead of a 15000B window
- **test_smallwindow_large_file**: like test_normal_large, but with a 500B window instead of a 15000B window

Some of you may see your FIN test failing. We set the timeout on that test a little too tight, and so in some cases the test timed out before your code was done sending FINs. In the notes by your grade, if it says "FIN test OK" or something similar, it means we examined your code and determined that you were handling FINs correctly that way.

Based on the results of these tests (and looking at your output, and in some cases your code), we assigned scores as detailed on the next page. We tried *really hard* to give partial credit whenever we could; this is why scores took so long to get back to you.

- Stop and Wait (50)
  - 0: Nothing works
  - 0.2 (10): RESETs and exits properly
  - 0.4 (20): Handles SYN or FIN, but not both, has problems completing the tests
  - 0.6 (30): Handles SYN and FIN, but has problems actually completing the tests
  - 0.8 (40): Handles neither SYN nor FIN, but still completes small and large tests
  - 0.9 (45): Handles SYN or FIN, but not both
  - 1 (50): Handles both SYN and FIN (test_syns and test_fins pass)
- Pipelining (20)
  - 0 : No pipelining at all
  - 0.25 (5): Sends way too much data, or resets after timing out on first packet
  - 0.5 (10): Sends way too little data, or a little more than too much (33 and above), but still attempts pipelining
  - 0.75 (15): Sends one too many packets (32) or one too few (28)
  - 1 (20): Succeeds (test_pipeline passes)
- Handle loss (15)
  - 0: Fails
  - 0.33 (5): Fails but simple mistake (close)
  - 1 (15): Succeeds (test_lossy succeeds)
- Variable window (10)
  - 0: Fails both small window tests (test_smallwindow and test_smallwindow_large_file)
  - 0.5 (5): one of the two tests above passes, but not the other
  - 1 (10): Passes both small and large file small window tests
- Reorder (10)
  - 0: Fails test_reorder
  - 1 (10): passes test_reorder
- Wraparound (5)
  - 0: for any error that appears in results due to wrong wrap around
  - 1 (5): if you transferred large files and handled sequence number wraparound satisfactorily