Project 2 Checklist (not necessarily in order of execution):

Overview:

* Client prompts for an HTTP request (GET TestFile.html HTTP/1.0)
* Testfile is stored in the Server host storage
* Server reads the test file
* Testfile is an ASCII file at least 80Kbytes in size
* Puts file into a buffer
* File is partitioned into 512 byte packets
* At end of file, 1 byte (NULL character) is transmitted to let the client know the file is ended.
* If the remaining data of the file is less than 512 bytes, the last packet will be padded with a NULL character.
* Each packet contains a header
* The header contains a checksum (for error detection) and a sequence number
* Server creates HTTP response message in 4 header lines, added on to the file being sent:

HTTP/1.0 200 Document Follows\r\n

Content-Type: text/plain\r\n

Content-Length: xxx\r\n

\r\n

Data

* Server will use the Go Back N protocol with ACK/NAK and retransmission (PAR)
* Window size is N=32
* Sequence number is mod 64
* After the server sends 32 packets, it will wait for an ACK/NAK from the client before continuing
* The client will print the sequence numbers, ACK/NAK, and data(48 bytes or so) (to show that the packets are sending and receiving correctly)
* The server will ALSO print sequence numbers and data sent, and ACK/NAK’s received... (this is implied, will email the prof/TA for clarification)
* The file will be closed after sending the NULL byte.
* After the file is closed, the packet is sent to an error detection function which takes the checksum from the header.
* When the packet is received by the client, the packet will be processed by the Gremlin function.
* After the packet is processed by the Gremlin and error detection, the packet is then processed into the segmentation and re-assembly function, to recreate the original file.
* After the file is fully received, the client-side should be able to display the reassembled HTML file.
* After the client receives a packet and verifies it is correct, it will send an ACK.

Go Back N Protocol

* The ACK/NAK will have a sequence number.
* If the error detection method finds an error, the client must send a NAK with the appropriate sequence number.
* If an error is discovered, a print statement should show that a NAK has been sent. (a bit redundant, but this part was clarified in BOLD)
* A packet that is received with errors should be dropped, and not passed to reassembly.
* If a packet is received with errors within a window, the rest of the packets within that window should be ignored until they are resent.
* If a packet is received out of order, the function that receives the packet must check the sequence number, and then print out the sequence number and a warning about lost/delayed packets.
* When the sender receives an ACK, it must advance the window accordingly (one at a time, I guess)
* When a NAK is received by the sender, the sender should resend the packets within the window starting with the sequence number of the erroneous packet.
* If an ACK/NAK is not received within a timeout window for any packet, the sender will resend all packets within that window starting from the timed out packet.
  + i.e. if packet 0 (of 31) is timed out, then all packets 0-31 will need to be resent.
* If an ACK is received within that time (after timeout and the sender is resending packets), then we can do one of two things:
  + Continue to resend the rest of the packets within the window before handling the ACK
  + Stop transmitting packets upon receiving the ACK, and move the window accordingly
* Set the timeout value to less than 20 milliseconds.
* The timeout value should not be more than 3 times the round trip time.
* All timers are reset once a NAK is received.
* If the last packet sent is less than 512 bytes, then pad the rest of it with 0’s.
* The packet header’s sequence number can be one byte long.

Gremlin:

* “When the program is executed” (asking for clarification), the user will input probabilities for damaged, lost, and delayed packets.
* The user will also input the length of the delay, in milliseconds.
* Functions for lost and damaged packets will be similar to that of Project 1.

Error Detection

* Error detection will also be similar to Project 1.

Testing

* Use the script command on the TUX machines to capture the execution trace of both the server and client. This trace file must contain the following (reiterating):
  + Whenever any packets are sent/received
  + Whenever any ACK/NAK’s are sent/received
  + Any print statements that packets were corrupted or lost
  + Sequence numbers (of packets and ACK/NAK’s)
  + “Other relevant information on the packets”

Submission

* Source codes
* Script
* Demo

`