Specs

* You only need to implement Go-Back-N
* You do not have to support outstanding connections (i.e., an incoming connection queue to support the listen backlog) in a passive open.
* You do not have to implement congestion control.
* You do not have to implement support for the URG and PSH flags, the urgent pointer, or urgent (out-of-band) data.
* You do not have to support TCP options.
* You do not have to implement a keep-alive timer
* You do not have to implement the Nagle algorithm.
* You do not have to implement delayed acknowledgements.
* You do not have to generate or handle ICMP errors.
* You may assume that simultaneous opens and closes do not occur
* You may assume that sock\_module only makes valid requests (that is, you do not have to worry about application errors)
* You may assume that exceptional conditions such as aborts do not occur.
* You should generate IP packets no larger than 576 bytes, and you should set your MSS (maximum [TCP] segment size) accordingly, to 536 bytes. Notice that this is the default MSS that TCP uses if there is no MSS option when a connection is negotiated.

Task List

1. Phase 1 – Complete
   1. Compile Minet
   2. Print arriving packets and socket requests
      1. Minet.h
      2. Packet.h
      3. Buffer.h
      4. Ip.h
      5. Tcp.h
   3. Implement MinetGetNextEvent’s timeout feature
   4. Implement connection FSM
2. Phase 2 – Data Flow
   1. Handle incoming ip packets by adding code to handle passive open
      1. minetAccept
      2. need to include a timer to handle lost packets
         1. tcp.h and ip.h helpful
      3. passive refers to when you receive a SYN in listen state
      4. need to send SYN-ACK and set a timeout for the expected ACK from the remote side
   2. Build out wrapper function framework for testing
      1. Variable for dropping probability
      2. Queue that is reordered to test packet reordering
   3. Implement Active opens with MinetConnect
      1. Need to send a syn to remote side
      2. Initialize tcp variables like sequence number and the send window and timeout
   4. Add code to handle data transfer
      1. **Figure out how timers are implemented**
      2. Implement flow control
         1. Recommended approach is stop-and-wait first then try go-back-n
   5. Handle active closes (Marc says you can do these with CTRL-D)
      1. Receive a FIN from the remote site or a close from the socket layer
         1. **Marc says it is not a FIN it is actually a WRITE with 0 bytes or something**
3. Phase 3 – Socket Layer
   1. Add code to keep track of outstanding requests in the active open state
   2. Add code to support connect request
      1. Create a connection address to state mapping and initiate an active open with connect
   3. Add code to support passive open
   4. Add code to ass passively open connection address up to sock module as zero byte WRITE requests
   5. Add close
   6. Add code to support WRITE with push data into the connection’s output queue
   7. Add code to send new data up to the sock module as WRITE requests
      1. Sock can refuse a write, should wait and resend later
   8. Verify generation and use of STATUS requests are correct
4. Phase 4 – testing
   1. Run ./start\_minet.sh “tcp\_server u <port>” and accept a connection with nc on another machine, receive and print data and handle CTRL-C on the nc client as passive close
   2. “tcp\_client u <host\_ip> <port>” should be able to connect to a listening nc server on another machine and send data to it and handle an active close with CTRL-D
   3. “http\_server1 u <port>” should be able to connect to it from telnet and handle a get request as in project 1
5. README
   1. Names of team members
   2. Brief specification of work undertaken by each member
   3. Any specific you may need to share