Vicky Tu
CPSC 433 Computer Networks
03/27/2016

Assignment 4 Report

EMULATOR

Terminal 1:
```
$ perl trawler.pl 8888 scripts/three.topo
Trawler awaiting fish...
Got port 10000: assigning addr: 0
Got port 10001: assigning addr: 1
```

Terminal 2:
```
$ perl fishnet.pl emulate localhost 8888 10000
Node 0: started
server 21 2
Node 0: server started, port = 21
SNode 0: time = 1460512570043 msec
Node 0: connection accepted
.:.:.:.:.:.:.:.:.:.:.&?&?.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:FNode 0: time
= 1460512663287 msec
Node 0: connection closed
Node 0: total bytes received = 50000
```

Terminal 3:
```
$ perl fishnet.pl emulate localhost 8888 10001
Node 1: started
transfer 0 21 40 50000
S:Node 1: time = 1460512570225 msec
```

```
Node 1: started
Node 1: bytes to send = 50000
.:..:.:..:.:.:..:.:::.?.?!.:..::...:.:.:..:.:.:.::::.:.....:..:.
:.:.::::::.:.......:.:.:.:::::::.:.........:.:::::::::.:........
..::::::::::.:..........::::::::::.:..........::::::::::.:......
....::::::::::.:..........::::::::::.:..........::::::::::.:....
......::::::::::.:..........::::::::::.:..........::::::::::.:..
........::::::::::.:..........::::::::::.:..........::::::::::.:
..........::::::::::.:..........::::::::::.:..........::::::::::
.:..........::::::::::.:..........::::::::::.:..........::::::::
::.:..........::::::::::.:..........::::::::::.:..........::::::
::::.:..........::::::::::.:..........::::::::::.:..........::::
::::::.:..........::::::::::.:..........::::::::::.:..........::
::::::::.:..........::::::::::.:..........::::::::::.:..........
::::::::::.:..........::::::::::.:..........::::::::::.:........
..::::::::::.:..........::::::::::.:..........::::::::::.:......
....::::::::::.:..........::::::::::.:..........::::::::::.:....
......::::::::::.:..........::::::::::.:......::::::Node 1: time
= 1460512662588
Node 1: sending completed
Node 1: closing connection...
FNode 1: time = 1460512663594 msec
Node 1: connection closed
Node 1: total bytes sent = 50000
Node 1: time elapsed = 93368 msec
Node 1: Bps = 535.5153800017137
```

Terminal 1 assigns addresses to Nodes 0 and 1 that want to connect to it. I started a server on Node 0 then transferred bytes to it from Node 1. The S in Terminal 3 means I sent SYN to the server. The . in Terminal 3 mean I sent a data packet, while in Terminal 2 mean I received a data packet. The . in Terminal 3 mean I received ACK, while in Terminal 2 mean I sent ACK. The F in Terminal 3 mean I sent FIN, while in Terminal 2 mean I received FIN.

SIMLATOR

Terminal 1:

```
$ perl fishnet.pl simulate 2 scripts/test.fish
Node 0: started
Node 1: started
Node 0: server started, port = 21
SSNode 0: time = 1005 msec
Node 0: connection accepted
:Node 1: time = 2010 msec
Node 1: started
Node 1: bytes to send = 50000
..::...::..::...::..::..::...::..::.::.:.:.&??.&??!.!..::...::.&
```

```
??!..::..!..::...::..!??.::..::..::..&??&??&??!.!..::..::...::..
::..::...&??&??&??!.!.!..::...::..::...::..::..::...::..::..::.:
.:..::&??&??!..::...::.::....::..:.::.....:.:.:::......::.::.::.
::.......::..::.::.::.::.:.:::.....::.::.::.::........::.::.::.:
:.::.:.:::.....::.::.::.::........::.::.::.::.::.:.:::.....::.::
.::.::.........::.::.::.::.::.:.:::.....::.::.::.::.........::.::.
::.::.::.:.:::.....::&??&??!..::...::.::.....::..::...::...::.::.:
.:::.....::.::.::!..::....::...::..::..::.::.:.:::......::&??&??!
..::...::.::.....::..::..::...:!.:....::.:.:!.:!??!??!?!???!?..::
...::..::.::....::...::...:!.:...::....:!.:...:!.:.!?!.?!?!.?!?!..
:!?!.!?!?:?!?!??!??!??!??!???..:!.!?:...::..::.::.....::..::...!..
::...:!.:...:!..:!.:.!??!??!?!.?!??!??!?!.!?.:?!?:!???...::...::
.:!.!.!?:..::....::..::.!..::...!.&??!??.:!.:...::..::!??!??!?!.
?!?.:?!?:?...::..::..::.::....::....:!.:...::.!..::...::.!??!.!??
!?!.!??!??!?!.?.:.::.:!??!??!?!???..::...:!.:...::!?!.?!??.:!.:!
??!??...:!.:...::.!.!??.:!.:...::..!?!.?!??!..:.:!..::.:!??!?!?!
???..:!.:..!?!.?.::...::!?!.?!??.:!.:!??...:!.:...::..!?!.?!??.:
!..::...::.!?!.?!??!?!.?.::.!??!.!?.:?!?:?...:!.:..!?!.?.:.:!.:.
!?!.?.:!?!.!?!.:!??..!.&??.:!.:..!??!..::...::..!?!.?!??!.&??&?!
..::..!?!.?.::...::!?!.?!?.:?!?:?..!.&??.:!.:..!??!..::...::!?!.
?!??.:!.:!??!??...:!.:..!?!.?.::...::.!?!.?!?.:!.:..!?!.?!??.:!.
:...::!?!.?!?!..:?!?:!???...:!..::..!??.:!..::!??....:!..::...::
..!?!.?!??.:!.:...::.!.!??!?!.?!??.:!..::.:!??!??!?!???..:!.!?!.
:..!?!.?.::...::!?!.?!??.:!.:!??!??...:!.:...::..!?!.?!??.:!.:..
.::..:!.:..!?!.?!?!.?!?!.?!?!.?.:!.:.!??!?.:!.?:!??!??!??!?!???.
!..:!.:..!?!.?.::...::!?!.?!??.:!.:!??!??...:!.:...::.!?!.?!??.:
!.:...:!?!.:..!??!?!.?.::..!??!?!.?.::.!??!?.:!.?:!?!??!???..:!.
:..!?!.?.::...::!?!.?!??.:!.:!??!??...:!.:...::.!?!.?!??.:!.:...
::..!?!.?!??!?!.?.::...:.::!?!.:!??!?!??!???..:!.:..!?!.?.::...:
:!?!.?!??.:!.:!??!??...:!.:...::.!?!.?!??.:!.:...::..!?!.?!??!?!
.?.::...:.:::!??!??!?!???..:!.:..!?!.?.::...::!?!.?!??.:!.:!??!?
?...:!.:...::..!?!.?!??.:!..:.:!.:..!??!?!.?!??.:!..::.:!??!?!??
?..:!.:..!?!.?.::...::!?!.?!??.:!.:!??!??...:!.:...::.!?!.?!??.:
!.:...::..!?!.?!??!?!.?.::...:.::!??!??!?!???..:!.:..!?!.?.::...
::!?!.?!??.:!.:!??!??...:!.:...::..!?!.?!??.:!.:...::..:!.:..!?!
.?!?!.?!?!.?!?!.?.:!.:.!??!?.:!.?:!??!??!??!??!??..:!.:..!?!.?.:
:...:!?!.:.!??.:!.:!?!??!??...:!.:...::..!?!.?!??.:!.:...::..:!.
:..!?!.!?!.?!?!.!?!.?.:!.:.!??.:!.!?:!??!??!?!???..:!.:..!?!.?.:
:...:!?!.:.!??.:!.:!??!??!??..:!.:..!?!.?.::...::..!?!.?.:!.:..!
.!??!?!.?.::...::!?!.?!??!?!..:?!?:!???...:!.:...::.!?!.?!?.:!.:
..::!??!??!??.!..:!.:..!?!.?.::..!?!.?.::...::..!?!.?!??.:!.:...
:.:::!??!??!?!???..:!.:..!?!.?.:.:!.:.!??!?!.?.::!??...:!.:...:!
?!.:..!??!.!?.:!.:...::..!?!.?!??!?!.?.::...:.:::!??!??!?!???..:
!.:..!..::...::!.!??.:!.:!??...:!.:...::..!?!.?.:!.:...::..:!.:.
.!?!.?!?!.?!?!.?!?!.?.:!.:.!??!?.:!.?:!??!??!??!??!?!??!??..:!.:
..!?!.?.::...::!?!.?!??.:!.!?:!??...:!.:...::.!?!.?!??!.&??.:!.:
```

```
..!??!?!..::...::.!?!.?!??.:!..::.:!??!?!???..:!.:..!?!.?.::...:
:!?!.?!??.:!.!?:!??...:!.:...::.!?!.?!??.:!.:...::..!?!.?!??!?!.
?.::...:.:::!??!??!?!???..:!.:..!?!.?.::...::!?!.?!??!..::!??...
:!.:...::..!?!.?!??!.&??&?!.?.::..!??!?!.?.::...:!?!.:..!??.:!.:
.!??!?.:!.?:!?!??!???..:!.:..!?!.?.::...::!?!.?!??.:!.:!??!??...
:!.:...::..!?!.?!??.:!.:...::..:!.:..!?!.?!?!.?!?!.?!?!..:!.:.!?
?!?.:!.?:!??!??!??!??!?!??!???..:!.:..!?!.?.::..!?!.?.::..::!??!
??...!.&??&?!.?.::..!??.:!.:...::.!?!.?!??.:!.:...::.!?!.?!??!?!
.?.:.::.:!??!?!???..:!.:..!?!.?.::...::!?!.?!??.:!.:!??!??...:!.
:...::.!?!.?.:!.:...::..!?!.?!??!?!.?.::...:.:::!??!??!?!???..:!
.:..!?!..::...::!?!.?!??.:!.:!??!??...:!.:...::..!..:!.:...::.!.
!??!?!.?!??.:!.:...::!?!.?!??!?!.?.:!?:!???...:!.:...::.!?!.?!??
.:!.:..:!?!.:!??!??!??..:!.:..!?!.?.::...::..!?!.?!??.:!.:...::.
.:!.:.!?!.?!?!.?!?!.?!?!..:?:!?!??!??!??!??!??..:!.:..!?!.?.::..
.::!?!.?!??.:!.:!??!??...:!.:...::.!?!.?!??.:!.:...::..!?!.!??!?
!.?.::...:.:::!??!??!?!???..:!.:..!?!.?.::...::!?!.?!??.:!.:!??!
??...:!.:...::..!?!.?!??!.&??&?!.?.::..!??!?!.?&?!.?.:!.:..!??!.
.::...::!?!.?!?.:?!?:?...:!.:...::.!?!.?!??.:!.:..::!??!??!??..:
!.:..!?!.?.::...::..!?!.?!??.:!.:...::..:!.:.!?!.?!?!.?!?!.?!?!.
.:?:!?!??!?!??!??!???..:!.:..!?!.?.::...::!?!.?!??.:!.:!??...:!.
:...::.!?!.?!?!.&??.:!.:..!?!.?.::...:!?!.:..!??.:!.:.!??!?.:!.?
:!?!??!???..:!.:..!?!.?.::...:!?!.:.!??.:!.:!??!??!??..:!.:..!?!
.?.::...::.!?!.?!??.:!..::.:!??!?!???Node 1: time = 95010
Node 1: sending completed
Node 1: closing connection...
FFNode 0: time = 96005 msec
Node 0: connection closed
Node 0: total bytes received = 50000
Node 1: time = 96010 msec
Node 1: connection closed
Node 1: total bytes sent = 50000
Node 1: time elapsed = 94000 msec
Node 1: Bps = 531.9148936170212
Fishnet exiting after time: 1000020 msec.
Number of packets sent: 2101
Number of packets dropped: 0
Number of packets lost: 117
```

I ran my simulator with to following fish file:

```
edge 0 1 lossRate 0.05 delay 5 bw 10000 bt 1000
time + 5
# server port backlog [servint workint sz]
0 server 21 2
time + 5
# transfer dest port localPort amount [interval sz]
1 transfer 0 21 40 50000
```

```
time + 1000000
time + 10
exit
```

In the printout we see a lot of ! followed by a sequence of . and : That means a retransmission of data, probably due to packet loss, was successful.

Brief Design Report:
I spent 20 hours simply understanding Fishnet and the assignment specification because I did not want to start down the wrong road. I talked to a bunch of people for 10 hours, and finally understood on a broad level what I need to implement. But the best part of this assignment were the 30 hours I spent pseudocoding with Amelia. We did it incrementally: stop-n-wait, fixed window size, flow control, then congestion control. We worked out a bunch of bugs on a whiteboard before they even became real bugs in our code. It was a lot of fun.

The assignment is to implement TCP using the Fishnet framework. That includes modify Node.java and writing TCPManager.java and TCPSock.java essentially from scratch. TCPSock.java implements the user interface of Java Sockets. Each TCPSock has a state (CLOSED, LISTEN, SYN_SENT, ESTABLISHED, SHUTDOWN), a type (WELCOME, WRITE, READ), a local port, a src addr and port, a dest addr and port, and a buffer. Welcome sockets need to be able to bind, backlog, and accept; write sockets need to be able to bind, connect, write, close, and release; read sockets need to be able to bind, processInput, read, and release.

The TCP Manager must keep a Hashmap of TCPSocks by src addr, src port, dest addr, dest port, and socket. While it could have sufficed to keep an array of sockets, I decided to use a Hashmap because I'm more familiar with the Java Hashmap class. The TCP Manager must also keep a list of ports and mark if they are being used. The default port for the reading sock is the port the server is listening on. It must have a queue of pending connections, since we want the first thing to request a connection to be the first one to get it.

To make this work, I had to make a slight modification to Node.java. In receivePacket, I made a switch statement that would redirect that packet to the TCP Manager if the protocol was TRANSPORT_PKT. While addTimer worked for timed out pings, I need to make it take more parameters for it to work as a Timer for Transport packets.

1. *Diss1a*: Your transport protocol implementation picks an initial sequence number when establishing a new connection. This might be 1, or it could be a random value. Which is better, and why?

   My transport protocol implementation picks a random initial sequence number when establish a new connection. This is more secure than using a constant value. Imagine the following scenario: I want to email Qiao details of the surprise birthday party we are planning for Yuchen. If I use 1 as my initial sequence number all the time, Yuchen could trick me into sending him the information by sending me ACK(seq = 2), so that I would connect and send the details to him. But if I always use a random initial sequence number x, Yuchen would not know x+1. When he sends me ACK(seq = x'+1), I would reject it. The random initial sequence number allows me to check the identity of the person I am connecting with.

2. *Diss1b*: Our connection setup protocol is vulnerable to the following attack. The attacker sends a large number of connection request (SYN) packets to a particular node, but never sends any data. (This is called a SYN flood.) What happens to your implementation if it were attacked in this way? How might you have designed the initial handshake protocol (or the protocol implementation) differently to be more robust to this attack?

   If I received many SYN(seq = x), my implementation would fill up the pending queue with pending connections. Then it would take many calls to accept before my listening socket could make a connection for a valid SYN request, since it needs to process everything in the pending queue before that valid SYN. To guard against this attack, I would want to start a timer with a short timeout for every entry I add to the pending queue. When the timeout is up, the timer would check if the entry has a socket in TCPManager's table, and if so, if the base number is higher than the SYN seqNum indicating data has been received from the client. If not, the socket should be told to close the connection, since the client is inactive.

3. *Diss1c*: What happens in your implementation when a sender transfers data but never closes a connection? (This is called a FIN attack.) How might we design the protocol differently to better handle this case?

   My implementation just waits if the sender does not close the connection. To make this better, I should set up a timer in the sock. Every time I get a DATA packet, I would record the time in a time-of-last-receipt field. After a timeout period, the timer would check if last time-of-last-receipt is the same as the current time-of-last-receipt. If yes, that means the client has not sent anything in a while, so my socket should send FIN, to initiate the close of the connection.

4. *Diss2*: Your transport protocol implementation picks the size of a buffer for received data that is used as part of flow control. How large should this buffer be, and why?

   By Little's Law, buffer size should be the average time bytes stay in the buffer times the rate at which packets are arrive. If we assume read is bring called pretty quickly as in TransferServer, our buffer need not be very large because the average time is pretty small.

1.1 In addition to a jar file, please provide a README to tell me what are examples you run and what is the expected result (screen capture would be fine.).

See the README in this directory.

1.2 Be sure to tell me both how to run your part1 and part2.

The program I submit implements both Part 1 and Part 2. If you'd like to see roughly how Part 1 functions allow, please use git to retreive that file.

1.3 Tell me how to compile your code should I have some problems running your jar file.

Type “make all”.

2. In your design document, be sure to include the following information in a separate section.

2.1 How TCP Manager, TCP Sock and Node behaves when you receive a packet and when you send a packet.

When I receive a packet that has protocol TRANSPORT_PKT, Node gives it to function receiveTCP in TCPManager.

If it is a SYN, I make a sockVals object from the packet, using all the information I need to make that socket the next time accept is called.

If it is a FIN, I close my socket and send an ACK back.

If it is a DATA, I check that it the packet I expect. If it is, I put the payload into my byte buffer, send an ACK, and advance my acknowledgement field. If the payload doesn’t fit in my bytebuffer, I pretend to have not received anything by not sending an ACK, not advancing the acknowledgement field, and not adding anything to my bytebuffer. If it isn’t, I send an ACK with the old ACK number.

If it is an ACK, I check if the seqNum is larger than my base number, If it is, while seqNum is greater than base, I remove the first packet off my window deque or transport deque, increment counter, and increment base, and repeat until the test condition if broken. Then I use counter to determine what my window size is, and I send number of packets equal to window size minus packets sent but not yet ACKed. I also calculate what the timeout time should be and set the window field.

When I send a packet, that’s using sendNext in TCPManager. This takes a packet off the top of my transport deque, records the send time, sends it, and puts it on bottom of my window deque. I then set a timer. When the timer is up, it calls function timeout in TCPSock. This checks if base is equal to seqNum, which means that this packet is holding up the line, so we push it back to the transport queue to resend. We also set window size back to 1.

2.2 How do you handle the sliding window in your code? (Brief description and locate the core code for me.)

I handle sliding window by using using 2 deques, transBuf and windowBuf.

(TCPSock.java, lines 71-72)

```
this.transBuf = new LinkedList<Card>();

this.windowBuf = new LinkedList<Card>();
```

transBuf holds all the packets that I've made, and is used by sendNext. windowBuf holds all the packets I've sent. When I receive a correct ACK, I removed from windowBuf.

(TCPManager.java, lines 134-141)

```
        while (seqNum > sock.base){

                /* remove any packet w lower seqNum */

                if (!sock.windowBuf.isEmpty()){

                    card = sock.windowBuf.removeFirst();

                }else{

                    card = sock.transBuf.removeFirst();

                }
```

The number of unACKed packets is always less than or equal to window size because when every I call sendNext, I only call it for window size - packets sent but not yet ACKed. Thus windowBuf always has size window size or less.

(TCPManager.java, lines 167-170)

```
            while (sock.ackWindow - sock.windowBuf.size() > 0 &&
!sock.transBuf.isEmpty()){

                /* send enough to fill up the ackWindow size */

                sendNext(sock);

            }
```

2.3 How do you handle flow control? (Same requirement as 2.2)

I put the amount of buffer space left in the bytebuffer of the read socket in the window field of each ACK packet.

(TCPSock.java, lines 330, 337)

```
this.window = this.byteBuf.remaining();
```

(TCPManager.java line 244)

```
                    this.sendSAF(da, dp, sa, sp, Transport.ACK, sock.seqNum, sock.window);
```

When the client socket gets the ACK pack, it sets the window field of the client.

(TCPManager.java, line 130)

```
sock.window = packet.getWindow();
```

When write is called, it will pick the minimum of len and window size and use that to create packets.This ensure than a fast sender will not overwhelm a client that has very little space left.

(TCPSock.java, lines 269-272)

```
    if (len > this.window){

        len = this.window;

        //tcpMan.node.logError("SET LEN TO " + this.window);

    }
```

2.4 How do you handle congestion control? (Same requirement as 2.2)

Window Size: When I get all my ACKs of one window back, I increase my window size by one. If I timeout, I set my window size to 1.

(TCPManager.java, lines 161-165)

```
                    while (sock.counter >= sock.ackWindow){
                        /* AIMD */
                        sock.counter -= sock.ackWindow;
                        sock.ackWindow++;
                    }
```

(TCPSock.java, lines 373)

```
        this.ackWindow = ackW;
```

Timeout Time: When I send a packet, I record the time I send it using function now in Manager. When I get an ACK for that packet, I calculate sampleRTT, estimatedRTT, devRTT, and timeoutTime. I used timeoutTime to set the time before the timer goes off for future packets.

(TCPManager.java, lines 309)

```
        curr.time = this.manager.now();
```

(TCPManager.java, lines 146-156)

```
                        if (card.dup == false && (sock.base == seqNum-len)){
                            /* calculate RTT on any non dup packets */
                            sock.samRTT = ackTime-card.time;
                            //System.out.format("ackTime: %d sendTime: %d%n", ackTime, card.time);
                            sock.estRTT = (long)(0.875*sock.estRTT) + (long)(0.125*sock.samRTT);
                            //System.out.format("estRTT: %d%n", sock.estRTT);
                            sock.devRTT = (long)(0.75*sock.devRTT) +
(long)(0.25*Math.abs((long)(sock.samRTT-sock.estRTT)));
                            //System.out.format("devRTT: %d%n", sock.devRTT);
                            sock.timeoutTime = sock.estRTT + (long) (4*sock.devRTT);
                            //System.out.format("Setting timeoutTime to %d%n", sock.timeoutTime);
                        }
```

Triple Duplicate ACK: I set a counter for the number of seqNums I get that are equal to my base. If this hits 3, I know I need to resend that packet b/c it probably got lost. I resend it using timer.

(TCPManager.java, lines 177-196)

```
}else if (seqNum == sock.base){

                System.out.print("?");
                sock.dup++;
                if (sock.dup == 3){
                    /* Triple Dup ACK */
                    int currW = sock.ackWindow/2;
                    if (!sock.windowBuf.isEmpty()){
                        Card card = sock.windowBuf.getFirst();
                        Transport tpkt = card.tpkt;
                        String[] paramTypes = {"TCPSock", "java.lang.Integer", "java.lang.Integer"};
                        Object[] objs = {sock, tpkt.getSeqNum(), currW};
                        /* resend immediately */
                        this.node.addTimer(0, "timeout", paramTypes, objs, sock);
                        //this.node.logError("Pkt " + tpkt.getSeqNum() + " got Triple Dup Acked;
resending now!!!!");
                    }
                    /* cut window size in half (redundant yes but I want you to see it) */
                    sock.ackWindow = currW;
                }
            }
```