Department of Computer Science
The Hong Kong University of Science and Technology

Student ID_____

COMP 361 Computer Communications Networks

Spring Semester 2004 - Midterm Examination – Solution key

Date March 25, 2004, Time 19:00 –21:00

Name:_____Student ID:_____Email:_____

Instructions:

1. This is a closed book exam
2. This examination paper consists of 15 pages and 8 questions
3. Please write your name, student ID and Email on this page.
4. For each subsequent page, please write your student ID at the top of the page in the space provided.
5. Please answer all the questions within the space provided on the examination paper. You may use the back of the pages for your rough work. Each question is on a separate page. This is for clarity and is not meant to imply that each question *requires* a full page answer. Many can be answered using only a few lines.
6. Please read each question very carefully and answer the question clearly and to the point. Make sure that your answers are neatly written, readable and legible.
7. Show all the steps you use in deriving your answer, where ever appropriate.
8. For each of the questions assume that the concepts are known to the graders. Concentrate on answering to the point what is asked. Do not define or describe the concepts unless specifically asked to do so.

Question	Points	Score
1	5	
2	15	
3	15	
4	10	
5	8	
6	12	
7	20	
8	15	
TOTAL	100	

1. Answer the following true/false questions by circling either T or F (5 points)

(a) HTTP servers listen for client requests on port 80 **T** F

(b) DNS runs on top of UDP **T** F

(c) HTTP uses out-of-band control T **F**

(d) Port 20 is used by FTP to transfer data **T** F

(e) The sender in the TCP protocols keeps an individual timer for each unacknowledged packet. T **F**

2. (15 pts)
Consider the client-server JAVA code studied in class. In this question you need to describe the different *types* of sockets used by that code. It is only necessary to *describe* the types; you do not have to remember what they are called in JAVA.

(a) First consider the TCP code.
How many different *types* of sockets does the server have?
What is/are the function(s) of this/these socket(s).
If there is more than one type of socket please explain the difference between the types.

How many different *types* of sockets does the client have.?
What is/are the function of this/these socket(s).
If there is more than one type of socket please explain the difference between the types.

(b) Now consider the UDP code. Answer the same questions as in (a) for the client and the server.

(a) *The TCP server has two different types of sockets.*
The first type of socket is a ***server socket*** *that sits waiting for a request from a client to set up a connection.*
The second type of socket is the interface to the connection stream established between the processes on the two machines. Data goes into the socket on one process and comes out the socket on the other side.

The TCP client has only one type of socket. This socket is the same as the second type of socket that the server has

(b) The UDP client and server both have only one type of socket.
This socket (the same for both) is a ***datagram*** **socket** through which datagram packets are sent and received.

3. (15 pts)
Consider the Selective-Repeat protocol studied in class. Suppose that the sequence #s available are 0,1,2,3,4,5,6 and the window size used by the protocol is N=4. Will the protocol work properly in ensuring reliable delivery of packets in the presence of lost and/or corrupted data/ACKS?

If yes, explain why it will work. If no, give an example as to a situation in which the protocol will make a mistake. You should assume that if (i) packet A is sent before packet B AND (ii) both packet A and packet B arrive properly, then A arrives before B

No, it does not work.

As an example suppose that
(i) The Sender sends packets 0,1,2,3
Sender window is [0,1,2,3]
(ii) The Receiver receives all of those packets and sends ACK0, ACK1, ACK2, ACK3.
After the ACKS the Receiver window is [4,5,6,0] where the 0 is the packet after packet 6.

There are now two different scenarios

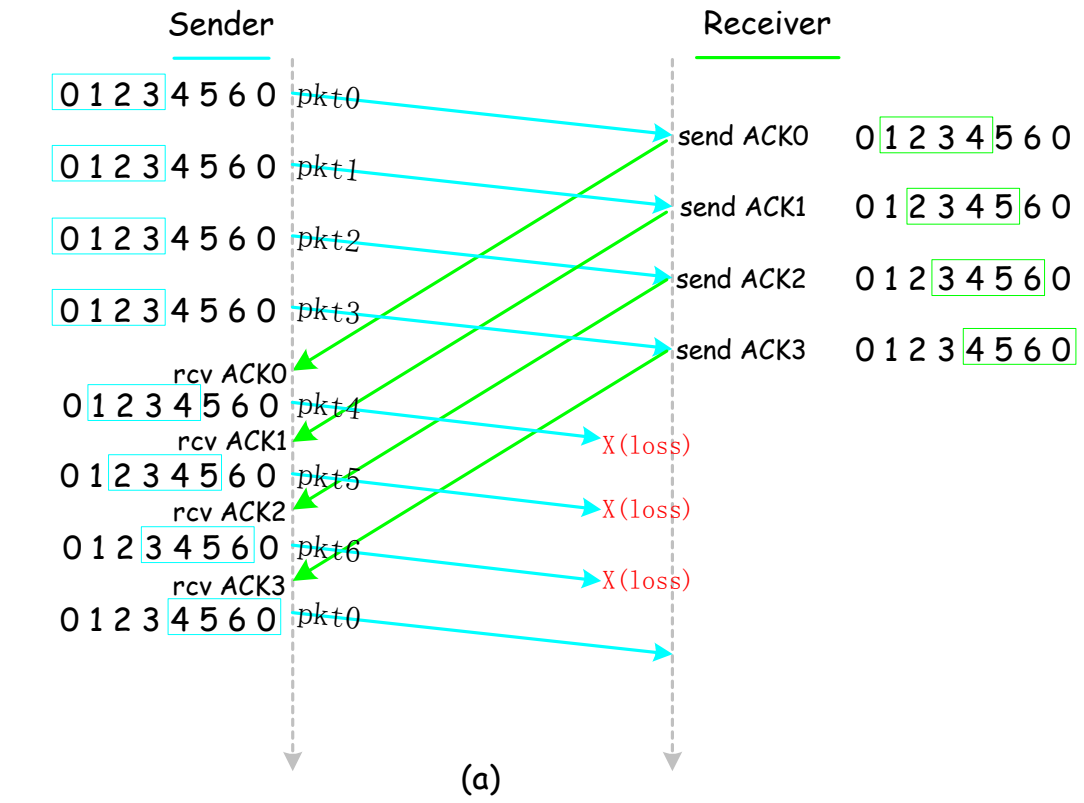
(a) ACK0, ACK1, ACK2, ACK3 all get through properly. As they arrive, sender sends out packets 4,5,6,0 where the 0 is a new packet after packet 6.

Packets 4,5,6 all get lost but
the new packet 0 is received by the receiver.

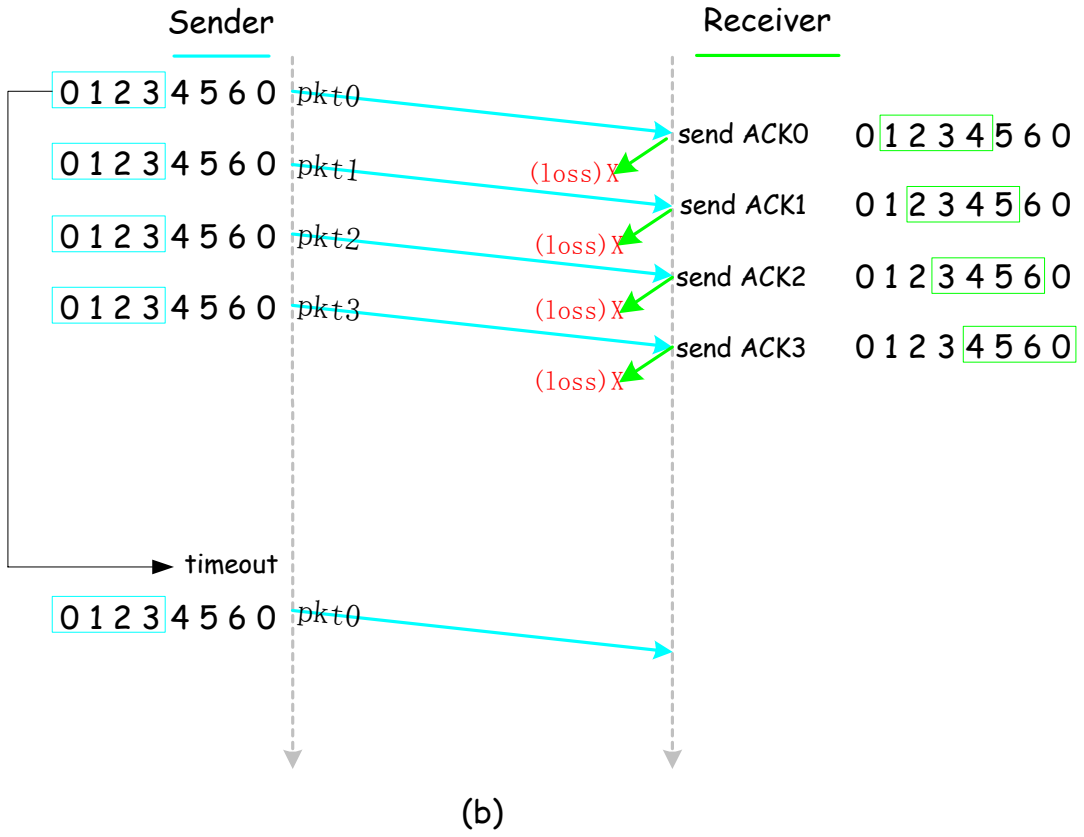
(b) ACK0, ACK1, ACK2, ACK3 all get lost.
Sender then times out on the original packet 0 and resends it.
Sender window is [0,1,2,3]
The resent packet 0 is then received by the receiver.

***Note that in both scenarios the receiver sees the same information; packets numbered 0,1,2,3,0, in that order**. Since it receives the same information in both scenarios, it is unable to distinguish between whether the last packet 0 that it received was a resend or a new packet. Therefore, the protocol does not work.*

The example above is illustrated on the next page.


Sender
Receiver
0 1 2 3 4 5 6 0
pkt0
pkt1
pkt2
pkt3
pkt4
pkt5
pkt6
send ACK0
send ACK1
send ACK2
send ACK3
0 1 2 3 4 5 6 0
rcv ACK0
rcv ACK1
rcv ACK2
rcv ACK3
X(loss)

(a)


Sender
Receiver
0 1 2 3 4 5 6 0
pkt0
pkt1
pkt2
pkt3
send ACK0
send ACK1
send ACK2
send ACK3
(loss)X
timeout

(b)

4. (10 pts)
In the TCP protocol what is meant by *fast retransmit*.
When is *fast retransmit* used?
What is the reason for using fast retransmit?

Fast Retransmit *means resending a packet before it times out.*

It is used when a sender receives three ACKS for the same data. In this case it retransmits the segment immediately after the ACK'd data.

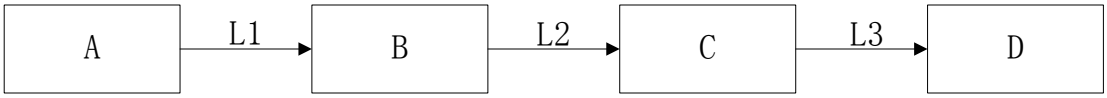
The reason for using fast retransmit is that the reception of 3 DUP ACKs is a strong signal that the next segment was lost. Since timeout periods are relatively long waiting for a full timeout period can lead to long end-to-end delay. Retransmitting before timeout when we have a strong probability of the packet being lost helps decrease this delay.

Note: to get full credit for this question you must specify WHICH packet is resent after the 3 Duplicate ACKS.

5. (8 pts)
You are given the three bytes below. Calculate their checksum (assuming that you are using 8 bit checksums) and write it in the appropriate space.

Byte 1	0 1 1 0 1 1 0 1
Byte 2	1 1 0 1 0 0 1 1
Byte 3	1 0 1 0 1 1 0 0
Checksum	**0 0 0 1 0 0 1 0**

6. (12 points)
Transmission/Propagation
Consider sending a file of **F=M*X** bits from A to D over the network below, consisting of four nodes and three links.



(a) First assume that the network is a packet-switched network and that the file is split into **M** packets each of size **X**. Each packet is then given an additional **h** bits of header. Each link transmits at **R** bps. Assume that propagation delay on the links is negligible

How much time does it take from when the first bit of the file leaves A to when the last bit arrives at D? Explain your answer.

(b) Next, suppose that the network a circuit-switched network. Further suppose that the transmission rate of the *circuit* between source A and destination D is **R** bps and that propagation delay on the links is negligible

Assuming t_s seconds set-up time to set up the circuit and **h** bits of header appended to the entire file, how long does it take from when the first bit of the file leaves A to when the last bit arrives at D? Explain your answer.

(a) (M+2)(X+h)/R

This is a special case of question 2(b) of the week 2 tutorial.

Each packet has (X+h) bits so the time to send it over one link is (X+h)/R.

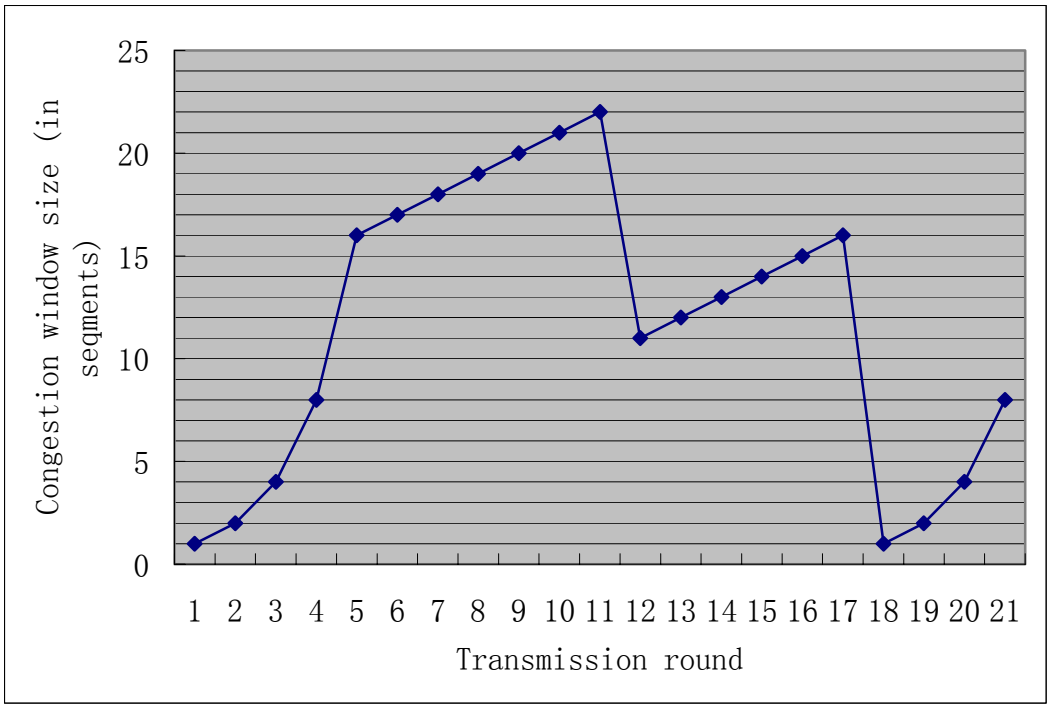
The M'th packet is completely sent over the first link after time M(X+h)/R, over the second link at time (M+1)(X+h)/R and over the third link at (M+2)(X+h)/R.

(b) t_s +(F+h)/R

Since there is only one header the total data sent is (F+h) bits. Since there are no packets and no stop and forward waiting the total time sent to transmit the data is (F+h)/R.

This is question 2(d) of the week 2 tutorial

7. (20 pts)
Consider the graph below modeling behavior experienced by a TCP protocol. The graph should be read to mean that **Congwin**=1 during round 1, **Congwin**=2 during round 2, **Congwin** =4 during round 3, etc..



Answer the following questions. In all cases give a short explanation as to how you derived your answer.

(a) Is this TCP RENO or TCP Tahoe?

(b) What is *slow-start*? In which rounds is slow-start operating?

(c) A loss event occurs during the 11th transmission round. What type of loss event occurs?

(d) A loss event occurs during the 17th transmission round. What type of loss event occurs?

(e) What is the initial value of **Threshold** during the first transmission round?

(f) Suppose there is no loss event during the 21st transmission round. What will the value of **Congwin** be during the 22nd round?

(a) RENO, since after the loss event in round 11 Congwin is set to Congwin/2 instead of 1.

(b) Slow start means setting Congwin=1 and then doubling the value of Congwin every round. Slow start is operating in rounds 1-5 and 18-21.

(c) 3 Dup ACKS. We know this because Congwin was only halved and not set to 1.

(d) A Timeout. We know this because Congwin was set to 1 and not halved.

(e) 16. We know this because slow start ends (and AIMD begins) when Congwin=16 (in round 5).

(f) 9. The last loss event was in round 17 when Congwin=16. At this point Threshold was set to be Congwin/2=8. Starting from round 18 through round 21 the protocol was in slowstart and did not change the value of threshold. In round 21 Congwin was set to be 8 which is the value of threshold so, in round 22, the protocol switches to AIMD and Congwin is only increased by 1, to 9.

8. (15 pts)
In this question you should explain how *Content Distribution Networks (CDNs)* use DNS to work.

Akamai is a company that operates *Content Distribution Networks* and sells its services to other companies. Assume that CNN.com has a contract with Akamai in which Akamai will distribute all of the JPEG files on CNN.com's web pages. All non-JPEG files will continue to be distributed directly by CNN.com. Explain how CNN.com and Akamai use DNS to redirect your requests for JPEG files on CNN's site so that the files are retrieved from an Akamai server `near' you.

A) You, through your browser, puts in a request for a web page on CNN.com. Your browser connects to CNN.com and requests the web page. If your browser (or proxy server) doesn't have the address of CNN.com cached it will need to get the address using DNS After getting the web page your browser checks the web page to see what extra files are referenced by the web page that it needs to request. Suppose the web page references file ***foo.jpg.***

B) When CNN.com provides the web page to your browser it replaces the reference *http://www.cnn.com/webpagefiles/foo.jpg* *with something like* *htpp://www.Akamai.com/www.cnn.com/webpagefiles/foo.jpg*

(note: the above is only an illustration of the fact that cnn.com is telling your browser that it should look for the picture foo.jpg at Akamai.com. The reality probably uses much more complicated URLS).

C) Your ***browser*** *now uses DNS to look up the address of Akamai.com, which is the hostname for the referenced file.*

Because of the way DNS works, all requests for Akamai.com addresses are sent to an authoritative name server for Akamai's domain (after going through the root name server).

Akamai's authoritative server checks where the request is coming from and responds, based on this information, *returning the IP address of an Akamai server which is `near' you. It can do this because Akamai controls its own authoritative servers and therefore can control how they respond to DNS requests. Akamai maintains its own internal map that lets it answer which of its servers are `near' a particular given IP address.*

D) Your browser then requests the jpeg file from the host at the returned IP address

The most important parts of this answer are the facts that (a) your browser requests the ***original*** *web page from cnn (b) cnn* ***relabels*** *the file so that you will have to ask Akamai for it (c)* ***your browser does the DNS search*** *for Akamai's address and Akamai's authoritative name server* ***responds to the request based on your browser's IP address*** *and then (d)* ***your browser requests the data*** *from the proper `nearby location.*