CS118 Midterm Exam, Fall 2010

Name: Solution

Student ID: _____

Notes:

1. This is a closed-book, closed-notes examination. But you can use the one-page cheat sheet.
2. You are allowed to use your calculator but not your computer.
3. Be **brief** and **concise** in your answers. Answer only within the space provided. If you need additional work sheets, use them but do NOT submit these sheets with the midterm examination.
4. Use the back pages for scratch paper. You should cross out your scratch work when you submit your exam paper.
5. If you wish to be considered for partial credit, show all your work.
6. Make sure that you have 9 pages (including this page and one-page Appendix) before you begin.

PROBLEM	MAX SCORE	YOUR SCORE
1	16	
2	24	
3	10	
4	10	
5	16	
6	24	
TOTAL	100	

DO NOT TURN TO THE NEXT PAGE UNLESS YOU GET PERMISSION !!

Problem 1: Multiple choices (2 points each). Select all the correct answers from the five choices.

1. Which of the following application-level protocol uses UDP?

   - Your answer ____ (A) HTTP; (B) FTP; (C) DNS; (D) SMTP; (E) BitTorrent.

2. Suppose TCP congestion window size is 10 segments, and its receiver's advertised window size is 8 segments. What is the maximum number of back-to-back packets TCP can transmit in its reliable data transfer?

   - Your answer ____ (A) 8; (B) 9; (C) 16; (D) 18; (E) 20.

3. Which protocol is NOT used when Bob uses his newly bought laptop computer to access his UCLA emails for the first time?

   - Your answer ____ (A) SMTP; (B) DNS; (C) POP3 or IMAP; (D) FTP; (E) TCP.

4. Which header field is NOT present in both UDP and TCP packet headers?

   - Your answer ____ (A) Source port number; (B) Destination port number; (C) Checksum; (D) Sequence number; (E) Acknowledgment number.

5. Which is not a feature of packet switching?

   - Your answer ____ (A) statistical multiplexing; (B) no reservation is needed in advance; (C) providing delay guaranteed services; (D) more efficient for bursty data traffic; (E) congestion may occur in the network.

6. Which mechanism is not required to ensure reliable data transfer?

   - Your answer ____ (A) error detection via checksum; (B) automatic error correction for corrupted packets; (C) retransmission upon timeout; (D) sequence numbers for transmitted packets; (E) acknowledgment numbers for received packets.

7. Which layers in the protocol stack are NOT typically implemented at routers?

   - Your answer ____ (A) application layer; (B) transport layer; (C) network layer; (D) link layer; (E) physical layer.

8. What are the bad effects without network congestion control?

   - Your answer ____ (A) packets may be dropped due to congestion; (B) packets may have to be retransmitted; (C) buffer overflows at the congested router; (D) increased delay for packet delivery; (E) packets will take different delivery paths to reach the destination.

Problem 2 (3 points each): Answer the following questions. Be brief and concise.

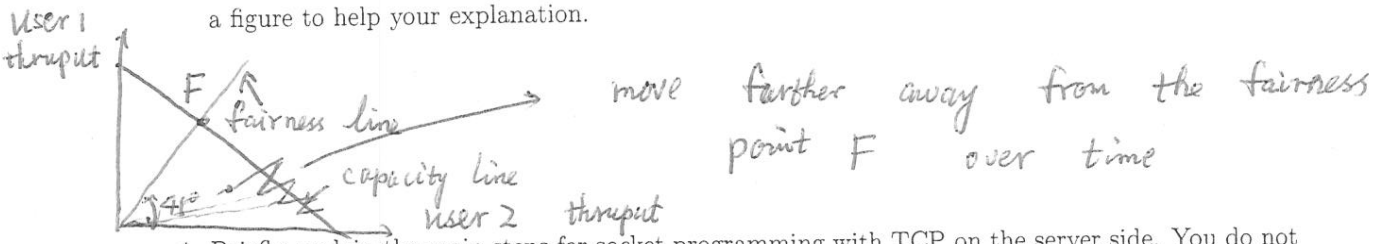
1. Web caching can reduce user-perceived response times, but cache may become stale over time. How does HTTP allow for a cache to verify that its objects are up to date?

"conditional GET"

2. A UDP receiver computes the Internet checksum for the received UDP segment and finds a *mismatch* with the value carried in the checksum field. Can the receiver be absolutely certain that bit errors have occurred with the received UDP data? Briefly justify your answer.

No. For example, when the checksum bits are corrupted but UDP data are correct.

3. Consider two TCP connections sharing a single link, with identical round-trip-times and segment size. It is well known that the additive-increase, multiplicative-decrease (AIMD) mode can ensure fair throughput for both TCP connections eventually. Now some one claims that multiplicative-increase, additive-decrease (MIAD) can also ensure fair throughput eventually for these two connections, starting from an arbitrary window size. Show why this is NOT true. You can draw a figure to help your explanation.



4. Briefly explain the main steps for socket programming with TCP on the server side. You do not need to list the detailed function calls.

1. open sockets, listen to well-known ports
2. accept connection request
3. transmit data
4. close TCP connection

5. Which HTTP operation model consumes the largest amount of server resources, nonpersistent but with parallel TCP connections, persistent connections with pipelining? Briefly justify your answer.

nonpersistent with parallel TCP connections, OS needs to allocate resources for each of the multiple parallel TCP connections

6. Specify two scenarios in reality where the local DNS server does not need to send the query to the root DNS servers.

1. local DNS server has cached query results
2. local DNS server has cached TLD server for the destination domain

7. A group of friends decide to develop a new Internet social networking application running on their own computers, which are on and off all the time. Which model the application should use, the client-server paradigm or the peer-to-peer one? Briefly justify your answer.

peer-to-peer, since each machine may not be on all the time, p2p works better

```
    n = write(sockfd, buffer, strlen(buffer));
    if (n < 0)
         error("ERROR sending with socket");
    bzero(buffer,256);

    n = read(sockfd, buffer, 255);
    if (n < 0)
         error("ERROR receiving from socket");
    printf("%s\n",buffer);
    return 0;
}

void error(char *msg) {
    perror(msg);
    exit(0);
}
```

Problem 4 (10 points): You are asked to compute the retransmission timeout (RTO) for TCP. The initial estimated round-trip time (RTT) is set as 400ms, and initial RTT variation is 200ms. The RTT samples for 4 TCP segments are 200ms, 400ms, 600ms, 400ms. In these 4 segments, the third TCP segment has been retransmitted once. Compute *all* four RTO values upon receiving *each* of four TCP segments. Show all the intermediate steps in your calculation. The following formula can be useful for your calculation:

$$EstimatedRTT = \frac{7}{8} \cdot EstimatedRTT + \frac{1}{8} \cdot SampleRTT$$

$$DevRTT = \frac{3}{4} \cdot DevRTT + \frac{1}{4} \cdot |SampleRTT - EstimatedRTT|$$

1st Segment: $EstRTT(1) = 7/8 \times 400 + 1/8 \times 200 = 375$

$DevRTT(1) = 3/4 \times 200 + 1/4 \times |200 - 375| = 193.75$

$RTO(1) = EstRTT(1) + 4 \times DevRTT(1) = 375 + 775 = 1150$ ms.

2nd Segment: $EstRTT(2) = 7/8 \times 375 + 1/8 \times 400 = 378.125$

$DevRTT(2) = 3/4 \times 193.75 + 1/4 \times |400 - 378.125| = 150.781$

$RTO(2) = EstRTT(2) + 4 \times DevRTT(2) = 981.249$ ms.

3cd segment: Ignore. We're not sure whether RTT is for the orginal or next trasmitted on.

$RTO(3)$ is the same as $RTO(2) = 981.249$ ms.

4th Segment: $EstRTT(4) = 7/8 \times 378.125 + 1/8 \times 400 = 380.85\ldots$

$DevRTT(4) = 3/4 \times 150.781 + 1/4 \times |400 - 380.85| = 117.9495$

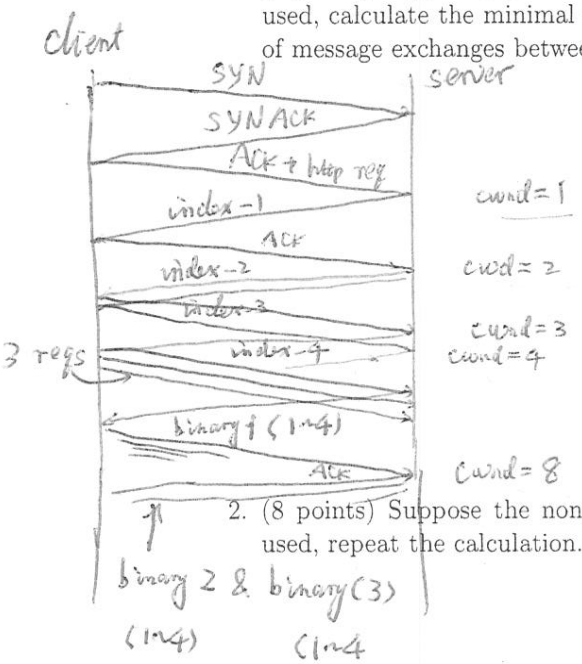
$RTO(4) = EstRTT(4) + 4 \times DevRTT(4) = 852.64$ ms

Problem 5 (16 points): A web browser running on the client host is requesting a webpage from the server. We make the following assumptions:

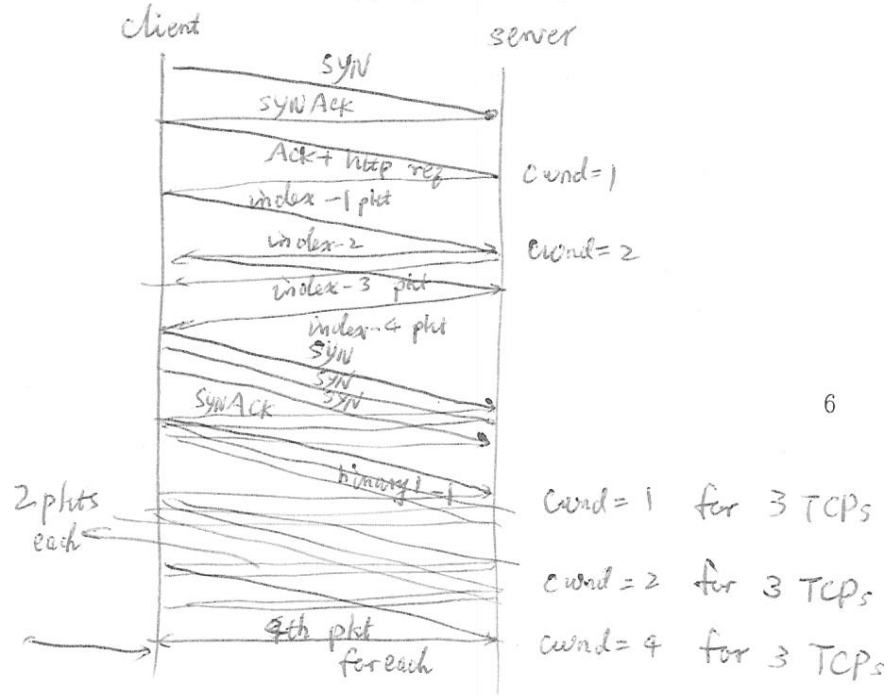
- TCP window size is 1 packet once the TCP handshake is complete. TCP header size is h bits, and the maximum payload size is p bits. Assume that the slow start threshold $ssthresh$ is set to 10 packets initially, and TCP uses its congestion control to adjust its window size.
- The link bandwidth is b bps, and the propagation delay is d seconds.
- Ignore DNS related delays, and ignore the payload (i.e., packet transmission delay) in three-way handshake packets, ACK packets, and HTTP request packets. In other words, those packets consist of header only. Each HTTP request is sent in one TCP packet.
- The client requests a webpage consisting of an HTML file that indexes 3 binary files on the same server. Each of the file is $4p$ bits long. In other words, each of the file can be sent in exactly 4 TCP packets. Piggybacking is used whenever possible.

Please answer the following questions:

1. (8 points) Suppose pipelining of HTTP requests is allowed and no parallel TCP connections are used, calculate the minimal time it takes the browser to receive all the files. Show the diagram of message exchanges between the server and the client to receive maximum partial credit.

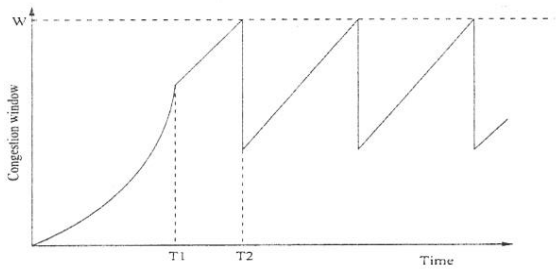


2. (8 points) Suppose the nonpersistent, non-pipelining mode with parallel TCP connections is used, repeat the calculation. Also show the diagram.



Problem 6 (24 points):

1. (9 points) Consider the following plot of TCP window size as a function of time.



- (2 points) Explain what happens in TCP congestion control during the interval $[0, T_1]$.

slow start

- (2 points) Explain what happens in TCP congestion control during the time interval $[T_1, T_2]$.

Congestion avoidance

- (5 points) Assume the TCP data packet size is MSS, and the TCP round-trip delay in steady state (i.e., after time T_2) is RTT. Please derive the loss rate (fraction of packets lost) of TCP for time after T_2. W is the maximum congestion window size defined in the number of packets, shown in the figure.

1 pkt is lost when the window grows from $\frac{W}{2}$ to W.

during congestion avoidance, cwnd increases by 1 pkt during each RTT.

Therefore, it takes $\left(\frac{W}{2}+1\right)$ RTTs)

The total number of pkts transmitted is

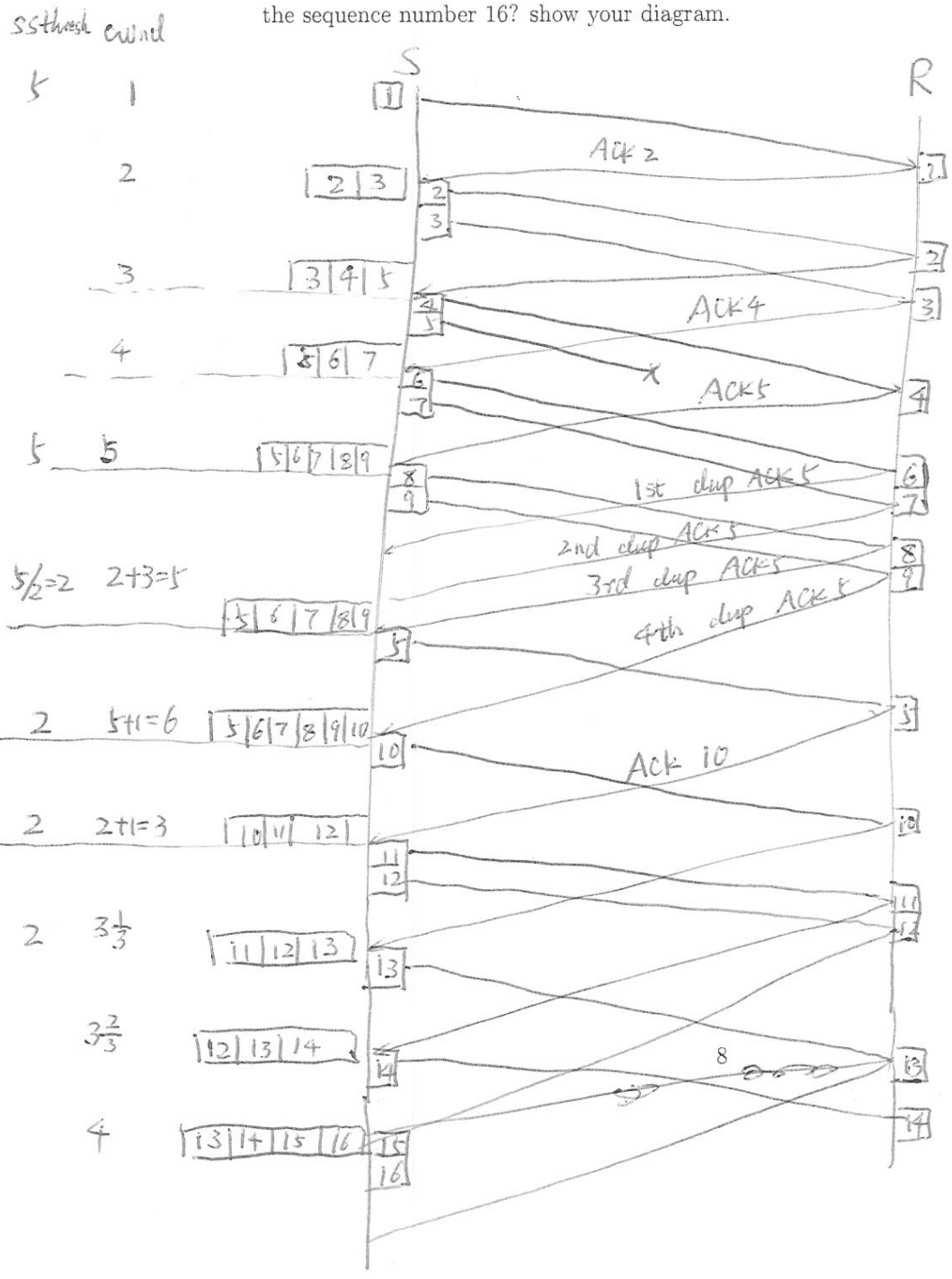
$$\frac{1}{2}\left(W+\frac{W}{2}\right)\times\left(\frac{W}{2}+1\right)=\frac{3}{8}W^2+\frac{3}{4}W$$

So, the loss rate will be: $\dfrac{1}{\frac{3}{8}W^2+\frac{3}{4}W}$

2. (15 points) Consider the evolution of a TCP connection with the following characteristics. Assume that all the following algorithms are implemented in TCP congestion control: slow start, congestions avoidance, fast retransmit and fast recovery, and retransmission upon timeout. If *ssthresh* equals to *cwnd*, use the slow start algorithm in your calculation.

- The receiver acknowledges every segment, and the sender always has data available for transmission.
- Initially *ssthresh* at the sender is set to 5, and *cwnd* as 1. Assume *cwnd* and *ssthresh* are measured in segments, and the transmission time for each segment is negligible. Retransmission timeout (RTO) is initially set to 500ms at the sender and is unchanged during the connection lifetime. The RTT is 100ms for all transmissions.
- The connection starts to transmit data at time t = 0, and the initial sequence number starts from 1. Segment with sequence number 5 is lost once. No other segments are lost.

What is the congestion window size cwnd when the sender starts to transmit the segment with the sequence number 16? show your diagram.



cwnd = 4
when transmitting
pkt #16

Appendix. Socket Programming Function Calls.

- *struct in_addr { in_addr_t s_addr; /* 32-bit IP addr */}*
- *struct sockaddr_in {*
  short sin_family; / e.g., AF_INET */*
  ushort sin_port; / TCP/UDP port */*
  struct in_addr; / IP address */ }*
- *struct hostent* gethostbyaddr (const char* addr, size_t len, int family)*
  struct hostent gethostbyname (const char* hostname);*
  char inet_ntoa (struct in_addr inaddr);*
  int gethostname (char name, size_t namelen);*
- *int socket (int family, int type, int protocol);*
  [*family*: AF_INET (IPv4), AF_INET6 (IPv6), AF_UNIX (Unix socket); *type*: SOCK_STREAM (TCP), SOCK_DGRAM (UDP); *protocol*: 0 (typically)]
- *int bind (int sockfd, struct socketaddr* myaddr, int addrlen);*
  [*sockfd*: socket file descriptor; *myaddr*: includes IP address and port number; *addrlen*: length of address structure=sizeof(struct sockaddr_in)]
  returns 0 on success, and sets *errno* on failure.
- *int sendto(int sockfd, char* buf, size_t nbytes, int flags, struct sockaddr* destaddr, int addrlen);*
  [*sockfd*: socket file descriptor; *buf*: data buffer; *nbytes*: number of bytes to try to read; *flags*: typically use 0; *destaddr*: IP addr and port of destination socket; *addrlen*: length of address structure==sizeof(struct sockaddr_in)]
  returns number of bytes written or -1. Also sets *errno* on failure.
- *int listen (int sockfd, int backlog);*
  [*sockfd*: socket file descriptor; *backlog*: bound on length of accepted connection queue]
  returns 0 on success, -1 and sets *errno* on failure.
- *int recvfrom (int sockfd, char* buf, size_t nbytes, int flags, struct sockaddr* srcaddr, int* addrlen);*
  [*sockfd*: socket file descriptor; *buf*: data buffer; *nbytes*: number of bytes to try to read; *flags*: typically use 0; *destaddr*: IP addr and port of destination socket; *addrlen*: length of address structure==sizeof(struct sockaddr_in)]
  returns number of bytes read or -1, also sets *errno* on failure.
- *int connect(int sockfd, struct sockaddr* servaddr, int addrlen);*
  [*sockfd*: socket file descriptor; *servaddr*: IP addr and port of the server; *addrlen*: length of address structure==sizeof(struct sockaddr_in)]
  returns 0 on success, -1 and sets *errno* on failure.
- *int close (int sockfd);*
  returns 0 on success, -1 and sets *errno* on failure.
- *int accept (int sockfd, struct sockaddr* cliaddr, int* addrlen);*
  [*sockfd*: socket file descriptor; *cliaddr*: IP addr and port of the client; *addrlen*: length of address structure==sizeof(struct sockaddr_in)]
  returns file descriptor or -1 sets *errno* on failure
- *int shutdown (int sockfd, int howto);*
  returns 0 on success, -1 and sets *errno* on failure.
- *int write(int sockfd, char* buf, size_t nbytes);*
  [*sockfd*: socket file descriptor; *buf*: data buffer; *nbytes*: number of bytes to try to write]
  returns number of bytes written or -1.
- *int read(int sockfd, char* buf, size_t nbytes);*
  [*sockfd*: socket file descriptor; *buf*: data buffer; *nbytes*: number of bytes to try to read]
  returns number of bytes read or -1.
- *int select(int maxfdp1, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *tvptr);*
  *FD_ZERO (fd_set *fdset);*
  *FD_SET (int fd, fd_set *fdset);*
  *FD_ISSET (int fd, fd_set *fdset);*
  *FD_CLR (int fd, fd_set *fdset);*