Sample Quiz for CS118

Notes:

1. <u>This sample quiz only illustrates the format of problems (i.e., multiple choices, short Q&A, and practical problem solving) to be tested. *It does NOT imply the scope, difficulty, weighting, or accuracy of the problems in the quiz.* In fact, this sample quiz covers all three chapters, but the upcoming Quiz 1 only covers Chapters 1 and 2.</u>
2. Therefore, do not spend time to work out these sample quiz problems; they will not appear in the quiz. Study the lecture notes, homeworks, projects, and textbooks for your preparation.
3. This is an open-book, open-notes quiz. You have about two hours to work on. Note that you need to upload your scanned copy or the photoed picture of your answer sheet to the Gradescope before the submission deadline. The deadline is strictly enforced, and no extension will be given.
4. You are allowed to use your calculator. You are advised not to use your Internet to search answers (hopefully you will not find much) during the quiz.
5. 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.
6. If you wish to be considered for partial credit, show all your work.
7. Make sure that you have 9 pages (including this page and one-page Appendix) before you begin.

PROBLEM	MAX SCORE	YOUR SCORE
1	12	
2	15	
3	8	
4	20	
5	20	
6	25	
TOTAL	100	

Problem 1: Multiple choices with justifications (2 points each). Select one answer from the given five choices, and justify your answer in *NO MORE THAN 20 words*.

1. Which of the following statement is true?
   - Your answer ____ (A) HTTP is a transport-layer protocol. (B) Modularity through protocol layering makes it easier to update system components. (C) POP3 is not an application-layer protocol. (D) SMTP uses UDP protocol at its transport layer. (E) DNS is not needed for the Internet.
   - Justification:

2. Which of the following statement on TCP is correct?
   - Your answer ____ (A) TCP provides the Internet's connection-less service. (B) TCP uses a 4-tuple of (source-IP-address, destination-IP-address, source-port, destination-port) as its identifier. (C) TCP implements congestion control but does not have flow control. (D) TCP does not implement reliable data transfer. (E) TCP operates at the network layer.
   - Justification:

3. How many unique sequence numbers do we need in the Selective Repeat protocol? Assume that the sender has a window size that stores 15 packets.
   - Your answer ____ (A) 2; (B) 15; (C) 16; (D) 30; (E) 31.
   - Justification:

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

1. How does the Web server (e.g., Amazon) identify users when you do the Internet shopping? Briefly explain how it works.

2. Explain how DNS uses recursive query to resolve a hostname translation.

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 additive-increase, additive-decrease (AIAD) 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 client side. You do not need to list the detailed function calls.

5. A group of users decide to develop a new Internet application running on their own computers, which are on and off all the time. Explain why the client-server model is not the right paradigm. What is the proper model to use to build this Internet application?

Problem 3 (8 points):

Joe is writing programs with a client and a server that use stream sockets. The following is the SERVER code that Joe wrote. Can you help Joe to find at least four errors in his code ? You can mark your answers in his code, and label the errors in the code. You can use the Appendix for references.

```
#include <server.h>
#define MYPORT 3490    /* the port users will be connecting to */
#define BACKLOG 10     /* how many pending connections queue will hold */
main()
{
    int sockfd, new_fd;  /* listen on sock_fd, new connection on new_fd */
    struct sockaddr_in my_addr;    /* my address information */
    struct sockaddr_in their_addr; /* connector's address information */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1); }

    my_addr.sin_family = AF_INET;         /* host byte order */
    my_addr.sin_port = htons(MYPORT);     /* short, network byte order */
    my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
```

```
bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the struct */

if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))== -1) {
    perror("bind");
    exit(1);  }

if (accept(sockfd, BACKLOG) == -1) {
    perror("accept");
    exit(1); }

while(1) {  /* main loop */
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = listen(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1){
        perror("listen");
        continue;  }
    printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));
    if (fork()) { /* this is the child process */
        if (sendto(new_fd, "Hello, world!\n", 14, 0) == -1)
            perror("sendto");
        close(new_fd);
        exit(0); }

    close(new_fd);  /* parent doesn't need this */
   while(waitpid(-1,NULL,WNOHANG) > 0); /* clean up child processes */ }}
```

Problem 4 (15 points): You are asked to compute the retransmission timeout (RTO) for TCP. The initial round-trip time (RTT) is set as 100ms. The RTT samples for 3 TCP segments are 200ms, 400ms, 250ms. In these 3 segments, the third TCP segment has been retransmitted twice. Compute *all* three RTO values upon receiving *each* of three TCP segments. Show all the intermediate steps in your calculation.

Problem 5 (25 points): A web browser running on the client host is requesting a webpage from the server. We make the following assumptions:

- TCP window is large once the TCP handshake is complete. TCP header size is h bits, and the maximum payload size is p bits.
- The bandwidth is b bps, and the propagation delay is d seconds.
- Ignore DNS related delays, and ignore the payload in three-way handshake packets, ACK packets, and HTTP request packets. In other words, those packets consist of header only.
- The client requests a webpage consisting of an HTML file that indexes 5 binary files on the same server. Each of the file is $2p$ bits long. In other words, each of the file can be sent in exactly 2 TCP packets. Piggybacking is used whenever possible.
- Each HTTP request is sent in one TCP packet.

Please answer the following questions:

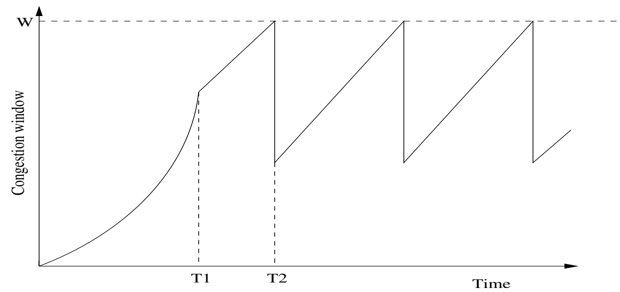
1. Suppose pipelining of HTTP requests is allowed and no parallel TCP connections are used, calculate the minimal time it takes the browser to receive all the files.

2. Suppose the nonpersistent, non-pipelining mode with parallel TCP connections is used, repeat the calculation.

3. Which mode gives the smaller latency? Briefly justify your answer.

Problem 6 (25 points):

1. Consider the following plot of TCP window size as a function of time.



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

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

   - Has retransmission timeout happened when packet loss occurs? Why does not happen?

   - 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 steady-state, average bandwidth of TCP for time after T_2.

2. 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 6. 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 4 is lost once. No other segments are lost.

   How long does it take, in milliseconds, for the sender to receive the ACK for the segment with the sequence number 12? show your intermediate steps or your diagram.

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)*;