Example Questions for Midterm
EE122, Fall 2008
EECS Berkeley

Note: The midterm exam will have six problems or more. The five problems below represent around 60-70% of the workload you should expect at the midterm exam.

Question 1. General Networking Concepts
For each of the following statements, indicate whether the statement is True or False, and provide a very short explanation of your selection.

a. Switches exhibit lower latency than routers. T F
   Rationale:

b. End-to-End Argument favors placing complexity at the lowest layers of the network stack. T F
   Rationale:

c. Packet switches have queues while circuit switches do not. T F
   Rationale:

d. “Best effort” means packets are delivered to destinations as fast as possible. T F
   Rationale:
Question 2. Network Performance Metrics

Complete each statement with a short “fill in the blank” phrase (1 point per fill-in):

a. File transfer, Email, and web access require two properties from the network:

1. ________________________________________________________________________________
2. ________________________________________________________________________________

b. Consider the network performance metrics Delay, Bandwidth, and Loss. For the following applications, which of the three is the most critical metric and why?

1. Distribution of web pages to remote web caches: _______________________________
   Rationale: ________________________________________________________________________

2. Two-way “Voice over IP” packet telephony: _______________________________
   Rationale: ________________________________________________________________________

3. Remote copying of large databases for disaster management: ____________________
   Rationale: ________________________________________________________________________
**Question 3.** Consider the network in the figure below where two source nodes A and B are connected to a
destination node D through a router C. Assume that node A starts to send a 600 bit packet at time 0 and
node B start to send a 1000 bit packet at time T (see figure below). Plot the inter-arrival time, denoted I,
between the two packets at node D versus the starting time of B’s packet, T for 0 <= T < 5.

Notes: Ignore the processing time at C. The arrival time of a packet is the time when the last bit of the
packet has arrived at node D. The inter-arrival time is

\[ I = (\text{arrival time of packet sent by B at D}) - (\text{arrival time of the packet sent by A at D}). \]

![Diagram of the network with nodes A, B, C, and D, and the time labels for sending packets.](image)
**Hint:** Based on the value of $T$, there are three cases you might want to consider. The diagram in Figure (a) shows the first case and depicts the messages sent by A and B arriving at C. You can use these diagrams to solve the problem, i.e., finish diagram (a) and fill in the other two diagrams (b) and (c).

(a) $T+6 \leq 8$

(b) $T+5$

(c) $T+6$

Use the following coordinates to plot the inter-arrival time between the two packets at D (I) versus the starting time of B’s packet (T), for $0 \leq T \leq 5$.

![Coordinate Plot](image-url)
Question 4. Assume two end-hosts communicate using the sliding window protocol. Assume the receiver window is always smaller than the sender’s window and the size of the receiver window is $w$ bits. Let $C$ be the link capacity between the two end-hosts in bps, and RTT be the roundtrip time between the two end-hosts in sec. What is the maximum throughput achieved by the two end-hosts?

Note: Assume every bit is acknowledged.
Question 5. An undergraduate named Stan Fjord took a network programming class last semester. His friend Berk Li is taking a similar class this semester. Stan decided to exhibit his supposedly superior knowledge by showing Berk how to write an echo server. This server just reads data from a client and sends it back unmodified to the same client. It is **required** to handle multiple clients concurrently by using the `select()` call to multiplex network I/O. It is quite similar to your project 1 client except that in addition, it **must** keep track of all the connected clients. It also sends a spiffy welcome message whenever a client connects. Unfortunately, Stan was dreaming in class when the professors talked about network programming pitfalls. Berk, who is obviously smarter, managed to find 3 major problems with the code. Can you match Berk and identify these problems? [Note: If the same kind of error appears in multiple places, then it still counts as one error only – but you should still report all instances]

All errors occur within the `while(1)` loop. Before this loop begins, assume that all variables are declared and initialized correctly; the `socket()`, `bind()`, and `listen()` calls are also done correctly. It is OK that no timeout is used in `select()`. Ignore syntax errors (misspellings, incorrect number/type of arguments) and performance issues. Focus on the correct operation of a server designed to handle multiple clients concurrently using only **blocking calls**. Also, think what might happen if a client doesn’t behave as it’s expected to.

```c
int main (int argc, char** argv) {
    /* declare and initialize all variables correctly */
    /* perform socket(), bind(), and listen() correctly */

    for(int i=0; i<FD_SETSIZE; i++) { /* initialize clients sockfd array */
        clients[i] = -1;
    }

    maxfd = listenfd;

    while (1) {
        FD_ZERO(&rfdset); /* set the fds we care about in rfdset */
        FD_SET(listenfd, &rfdset);
        for (int i=0; i<MAXCLIENTS; i++)
            if (clients[i] > 0)
                FD_SET(clients[i], &rfdset)
        select(maxfd+1, &rfdset, NULL, NULL, NULL);

        if (FD_ISSET(listenfd, &rfdset)) { /* check if need to accept new connection */
            connfd = accept(listenfd, (struct sockaddr*)&client_addr, &cli_len);

            for (int i=0; i<MAXCLIENTS; i++) { /* keep track of clients' sockfds */
                if (clients[i] < 0) {
                    clients[i] = connfd;
                    break;
                }
            }

            read(connfd, buf, MAXLINE); /* read from newly connected client */
            write(connfd, welcome_msg, strlen(welcome_msg)); /* send welcome msg */
            write(connfd, buf, MAXLINE); /* echo back to client */
        }

        /* for every readable connected client, read and write back, closing client connection */
        /* if necessary – this happens correctly so don’t worry about it*/
    }

    /* end of while(1) */
}
/* end of main() */
```