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: No routing table look-up, no delays associated with storing data ("queuing"), bits flow through the switch essentially as soon as they arrive.

b. End-to-End Argument favors placing complexity at the lowest layers of the network stack. T F
Rationale: Argues for placing complexity outside of the network and into the applications.

c. Packet switches have queues while circuit switches do not. T F
Rationale: Routers have queues; switches do not, even though the packet switch must have more memory than a circuit switch to receive a full packet before it can forward it on.

d. “Best effort” means packets are delivered to destinations as fast as possible. T F
Rationale: Refers to no guarantees about performance of any kind, not high performance.
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. **In order packet delivery**

2. **Reliable delivery**

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: **Loss**
   
   Rationale: *Low loss insures rapid dissemination of small files through the network;*

2. Two-way “Voice over IP” packet telephony: **Delay**
   
   Rationale: *Interactive responsiveness is needed, audio streams are loss tolerant;*

3. Remote copying of large databases for disaster management: **Bandwidth**
   
   Rationale: *Must move large bulks of data as quickly as possible; time to last byte is essential;*
**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

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

![Network Diagram](image-url)
Solution:

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 <= T <= 5.

I = T+8+d – (T+9.2+d) = -1.2ms  
I = 11.2+d – (9.2+d) = 2ms  
I = T+8+d – (9.2+d) = T-1.2 ms

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 <= T <= 5.

I (ms)  

T (ms)
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.

Solution: There are two cases (see figure below)

- case a: $RTT > w/C$, throughput = $w/RTT$
- case b: $RTT \leq w/C$, throughput = $C$
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);
        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() */
```
Solution

The errors are:

1. Server does a read() right after the accept(). If the client doesn’t send anything then server will be blocked and unable to accept/multiplex connections from other clients – fix: let select() detect if the newly accepted client’s is readable and only then read from it.

2. The return values of select(), accept(), read(), or write() calls are not checked at all. Need to do this to check for error and then take the appropriate action if necessary. Return value of read is necessary to give the right argument to the echo write.

3. maxfd is not updated when new connections come in. So even if we accept new connections select() won’t be able to check to see if they are readable (despite keeping rfdset up to date).

4. For incoming connections, there is no check to see if the server has exceeded MAXCLIENTS, in which case those connections should be rejected.