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

1. Suppose TCP congestion window size is 12 segments, and its receiver's advertised window size is 10 segments. What is the maximum number of back-to-back packets TCP can transmit in its reliable data transfer?
   - Your answer A (A) 10; (B) 11; (C) 12; (D) 20; (E) 22.
2. Which of the following protocol uses TCP?
   - Your answer A B D (A) HTTP; (B) FTP; (C) DNS; (D) SMTP; (E) BitTorrent.
3. Joe has a UCLA CS account and reads his emails from this account via outlook. Which protocols are used when he accesses emails sent by a friend bob@cs.stanford.edu?
   - Your answer C (A) DNS, FTP; (B) HTTP, FTP; (C) SMTP, DNS, IMAP/POP3; (D) FTP; (E) DNS, BitTorrent.
4. Which header field does appear in one but not both UDP and TCP packet headers?
   - Your answer D, E (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 C (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 B (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 of the following statement about DNS is wrong?
   - Your answer A (A) A local DNS server never queries the root DNS server; (B) DNS caching is used to improve performance; (C) Some of DNS queries can be iterative and others recursive, in the sequence of queries to translate a hostname; (D) DNS follows hierarchical design approach; (E) DNS do not use large centralized database.
8. Which layers in the protocol stack are NOT typically implemented at routers?
   - Your answer A, B (A) application layer; (B) transport layer; (C) network layer; (D) link layer; (E) physical layer.

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

3

1. Consider the queuing delay in a router buffer (preceding an outbound link). Suppose all packets are L bits, the transmission rate is R bps, and that N packets simultaneously arrive at the buffer every LN/R seconds. Find the average queuing delay of a packet.

   1st packet 0
   2nd pack L/R
   nth packet $(n-1)L/R$

   $\sum_{n=1}^{N} \frac{(n-1)L}{R}$

   Σ 0-n packets $= \Sigma\, 0 + \frac{L}{R} + \frac{2L}{R} + \ldots \frac{(n-1)L}{R} = \frac{\frac{L}{R}\frac{N(N-1)}{2}}{N}$ ← avg

   $= \boxed{\frac{L(N-1)}{2R}}$

0

2. Briefly explain how traceroute, which is based on ICMP (Internet control message protocol), works.

Traceroute works by tracing through the route of an IP request. It follows the path and helps in errors for the IP protocol

3

3. How does SMTP mark the end of a message body? How about HTTP? Can HTTP use the same method as SMTP to mark the end of a message body? Briefly justify your answer.

SMTP uses a line with a period on it and that's it to mark the end of a message body. HTTP uses the "content-length" header to know where the body ends. HTTP cannot use the same method as SMTP because binary data could be carried by HTTP which wouldn't work with a period at the end of the message.

3

4. 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, the checksum field itself could have been corrupted and the payload could be intact, but most likely the data has an error.

3

5. 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.

While AIMD converges towards the fairness/throughput cross, MIAD does not (it diverges). MIAD does not lead to fair throughput because the multiplicative increase sends the line away from the fairness point.

2

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

listen()

First a socket must be established and bound to, then we loop over the accept function, which accepts connections from the client side. Once a connection is made we can read/write to the connection, and eventually close the connection.

1

7. 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.

Non persistent w/ parallel TCP connections uses more server resources because of the overhead for all of the TCP connections that need to be created. In persistent connections only 1 TCP connection needs to be established, reducing the load on the server.

8. Which offers more scalable file distributions for a large number of users, the client-server model or the peer-to-peer model? Briefly justify your answer.

The P2P model offers more scalable file distributions because then the users can be servers and clients at the same time. At scale, this means that there is no need for large server banks to power large file sharing organizations, all that is needed is a few users that is willing to be seeders sharing.

9. Describe the step-by-step operation on both the client and the server when the client wants to close a TCP connection.

First the client sends a FIN to the server, which then ACKs that request. Then another FIN is sent, for which the client sends an ACK. Upon receiving the FIN, the client enters a timed wait before closing the connection fully

Problem 3 (8 points):

Joe is writing programs with a client and a server that use TCP stream sockets. The following is the CLIENT code that Joe wrote. Can you help Joe to fill in the, missing parts in his code ? You can use the Appendix for references.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[]) {
   int sockfd, portno, n;
   struct sockaddr_in serv_addr;
   struct hostent *server;

   char buffer[256];
   if (argc < 3) {
      fprintf(stderr,"usage %s hostname port\n", argv[0]);
      exit(0);
   }
   portno = atoi(argv[2]);

   sockfd = socket(AF_INET,SOCK_STREAM,0);
   if (sockfd < 0)
      error("ERROR opening socket");
   server = gethostbyname(argv[1]);
   if (server == NULL) {
      fprintf(stderr,"ERROR, no such host\n");
      exit(0);
   }
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family = AF_INET;
   bcopy((char *)server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
```

```
    serv_addr.sin_port = htons(portno);

    if (sockfd, (struct sockaddr_in *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting");

    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);

    n = sendto(sockfd, buffer, 256, 0, (struct sockaddr_in *) &serv_addr, sizeof(serv_addr));
    if (n < 0)
         error("ERROR sending with socket");
    bzero(buffer,256);

    n = recvfrom(sockfd, buffer, 256, 0, (struct sockaddr_in *) &serv_addr, sizeof(serv_addr));
    if (n < 0)
         error("ERROR receiving from socket");
    printf("%s\n",buffer);
    return 0;
}

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

4

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 80ms, and initial RTT variation is 40ms. The RTT samples for 3 TCP segments are 160ms, 120ms, 200ms. In these 3 segments, the 2nd TCP segment has been retransmitted once. Compute *all* RTO values upon receiving each of three TCP segments. Show all the intermediate steps in your calculation. The following formula can be useful for your calculation:

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

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

$$RTO = EstimatedRTT + 4 \times DevRTT$$

3

packet (1) $EstRTT = \frac{7}{8}(80) + \frac{1}{8}(160) = 70 + 20 = 90$ ms

$DevRTT = \frac{3}{4}(40) + \frac{1}{4}|160 - 80| = 30 + 20 = 50$ ms

$RTO = 90 + 4 \cdot 50 = 290$ ms

(2) since it was retransmitted, RTO stays the same

RTO = 290ms

(3) $EstRTT = \frac{7}{8}(90) + \frac{1}{8}(200) = 78.75 + 25 \approx 104$ ms

$DevRTT = \frac{3}{4}(50) + \frac{1}{4}|200 - 90| = 37.5 + 27.5 = 65$ ms

$RTO = 104 + 4 \cdot 65 = 364$ ms

Problem 5 (16 points): Consider sending 4 packets from Node A to Node C via Node B (see the figure below). The packet length is 500 bytes each. The propagation delay of both Link A-B and link B-C is 1 msec (0.001 second). Link A-B's bandwidth is 1Mbps (1×10^6 bits per second), and link B-C's bandwidth is 2Mbps.

A — 1 Mbps — B — 2 Mbps — C

Assume A starts transmitting the first packet at time t = 0.

1. What is the time gap between the first and second packets when they arrive at C? (i.e. the time gap between receiving the last bit of the first packet and the last bit of 2nd packet)

$$\left(\frac{500 \cdot 8 \text{ bits}}{10^6 \text{ bps}}_{A \to B} = \frac{4000}{10^6} = \frac{4}{1000} = 0.004\text{ s}\right) + \frac{500 \cdot 8 \text{ bits}}{2 \cdot 10^6 \text{ bps}}_{B \to C} = 0.002\text{ s}$$

Therefore, the amount of time between the first and second packets when they arrive at C is 4 MS because it takes 4 MS to clear the first link then 4 MS to clear the second by the time packet 1 has passed the B-C packet 2 is already in A→B. Therefore the time delay between the two is 4 MS.

16

2. When will C receive all the 4 packets?

1st packet takes 8 MS to get all the way through

2nd packet starts @ t=4MS, takes 8 MS arrives @ t=12

3rd packet starts @ t=8ms, arrives @ t=16

4th packet starts @ t=12ms, arrives @ t=20ms

total time to receive all 4 packets is 20 ms

Problem 6 (23 points):

1. Reliable transfer protocols

   - (3 points) In the Go-back-N Protocol, how does the sender react when timeout occurs?

     When timeout occurs, the GBN protocol will resend all packets that have yet to be ACKed for, and since they are cumulative ACKs, will usually be multiple packets

   - (3 points) In the Go-back-N Protocol, are 3 sequence numbers enough for the sender with a window size of 3 packets? Briefly justify your answer.

     No, because there needs to be enough sequence numbers to allow for [illegible] and retransmission, which 3 seq numbers will not be enough. Since GBN uses cumulative ACKs, if the window slides while a packet has not be been ACKed yet there needs to be more than 3 seq numbers.

   - (2 points) In the Select-Repeat protocol, how does the sender react when an ACK outside the window [send base, send base+window size] is received?

     If the ACK is less than send-base, then it is ignored because it must have been lost or something in the network, while if the ACK is greater than send base + window size, then the ACK is also ignored.

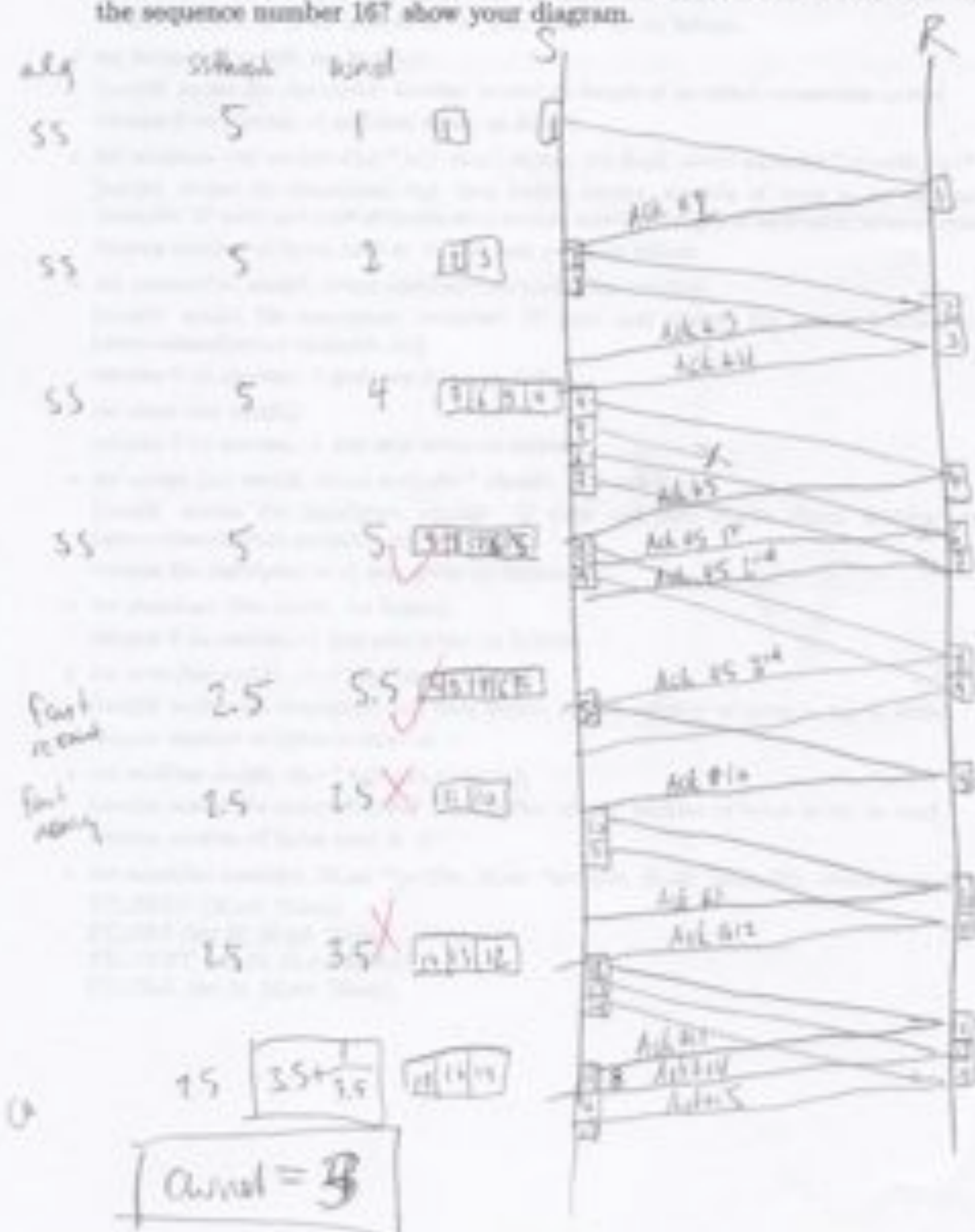
     The ACK will be ignored, because each packet sends back exactly one ACK, and if that ACK is outside the current window, then it is not important to the protocol

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. Right after fast retransmit/fast recovery phase, if *ssthresh* equals to *cwnd*, use the slow start algorithm.

- 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.
- Assume that the sender uses the following equation to update cwnd during congestion avoidance.

$$cwnd+ = MSS * MSS/\lfloor cwnd \rfloor$$

What is the congestion window size cwnd when the sender starts to transmit the segment with the sequence number 16? show 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);