CS118 Midterm Exam, Spring 2019

Name: [redacted]

Student ID: [redacted]

Notes:

1. This is a closed-book, closed-notes examination. But you can use the one-page cheat sheet.
2. You are not allowed to use your electronic devices.
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. Any answers or work shown on the back of any page will NOT be considered or graded.
5. If you wish to be considered for partial credit, show all your work.
6. Make sure that you have 10 pages (including this page and one-page Appendix) before you begin.

PROBLEM	MAX SCORE	YOUR SCORE
1	18	
2	18	
3	8	
4	12	
5	12	
6	12	
7	20	
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 four choices. Note that there can be *multiple* correct answers.

For your reference, the following are the full names for the used Acronyms: HTTP: hypertext transfer protocol; DNS: Domain name system; SMTP: Simple mail transfer protocol; POP3: Post office protocol version 3; IMAP: Internet Mail access protocol; TCP: transmission control protocol; UDP: User datagram protocol.

1. What would be the correct feature(s) of a file-distribution application using the peer-to-peer (P2P) model rather than the client-server model?
   - Your answer D (A) All peers have to remain always on; (B) A peer cannot leave even after its file downloading completes; (C) P2P model scales worse than the client-server model; (D) A peer can act as a client by issuing requests while serving other peers.
2. Which protocol is not used when Bob uses his smartphone to browse the ESPN Web site?
   - Your answer C (A) TCP; (B) HTTP; (C) IMAP; (D) DNS.
3. Which field(s) in the UDP header are used in connectionless demultiplexing?
   - Your answer B (A) Source port number; (B) Destination port number; (C) Length; (D) Sequence number.
4. What is true regarding Dynamic Adaptive Streaming over HTTP (DASH) for video streaming?
   - Your answer A, B (A) The video is encoded into several different versions, which have different qualities but have the same bit rate; (B) It allows clients with different Internet access rates to stream in video; (C) The client selects different chunks one at a time with HTTP PUT request messages; (D) It does not allow a client to dynamically adjust its streaming rate once the streaming session starts.
5. Which is a benefit of packet switching but not circuit switching?
   - Your answer D (A) congestion may occur inside the network; (B) reservation is always needed before data delivery; (C) providing delay guaranteed services; (D) statistical multiplexing
6. When can the first TCP data segment starts its transmission during the TCP connection?
   - Your answer C (A) together with the first SYN message; (B) before the second SYN+ACK message is received; (C) together with the third ACK message; (D) After the third ACK message has arrived at the receiver.
7. Which of the following statement about DNS is true?
   - Your answer B, C (A) A local DNS server never queries the root DNS server; (B) DNS uses caching to improve performance; (C) Some of DNS queries can be iterative and others recursive, in the sequence of queries to translate a hostname; (D) Only authoritative DNS servers can respond to queries.
8. Which layers in the protocol stack are implemented at the end host?
   - Your answer B (A) application layer, transport layer, network layer, link layer, physical layer; (B) application layer, transport layer; (C) network layer, link layer, physical layer; (D) application layer only.

9. Which mechanism of HTTP is used to allow a cache to verify that its objects are up to date?

   - Your answer C . (A) Cookies; (B) stateless HTTP server; (C) conditional GET; (D) HTTP with persistent connections.

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

1. In an ongoing TCP connection, the TCP sender receives a new Acknowledgment segment, which has its 'Receive Window' field set as 11000 Bytes. Before receiving this Acknowledgment, the TCP sender has not perceived any segment loss with its *cwnd* being 10000 Bytes and its *ssthresh* as 8000 Bytes. What is the window size updated by TCP in its reliable transfer right after receiving this Acknowledgment? Assume the maximum segment size (MSS) is 1000 Bytes. Show your steps.

   upon successful ACK receipt, $cwnd = cwnd + \left(\frac{MSS}{cwnd}\right)$
   since cwnd > ssthresh, the connection is in congestion avoidance, so
   $cwnd = 10{,}000 + \frac{1000}{10} = 10{,}100$ bytes
   window = min(cwnd, rwnd) so window size is 10,100 bytes

2. 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 all N packets simultaneously arrive at the buffer at time t=0. Find the queuing delay of the packet that is transmitted last.

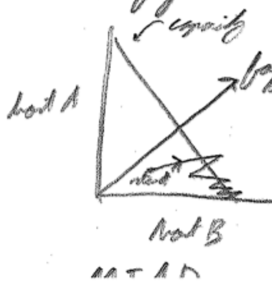
   # of bits preceeding the last packet $= (N-1) \cdot L$
   outgoing rate $= R$ bits/second
   total delay until starting to transmit the last packet $= \frac{(N-1)L}{R}$ seconds

3. A new video streaming startup *v-start* decides to use the third-party CDN provider *limelight* to scale and reduce streaming latency. Explain how to leverage DNS to intercept and redirect the user clicks on `http://video.v-start.com/` to the CDN servers deployed by *limelight*.

   The service sets up the DNS server for v-start.com to point to a CDN, so when a DNS query reaches video.v-start.com, it gives the IP of a CDN server nearby that serves the content directly

4. Consider two TCP connections sharing a single link, with identical round-trip-times and segment sizes. The additive-increase, multiplicative-decrease (AIMD) mode is known to ensure fair share for both connections eventually. Now a programmer decides to use multiplicative-increase, additive-decrease (MIAD) mode in his implementation. Explain whether this change will result in starvation of a connection (i.e., if starting from an arbitrary window size, one connection will eventually be starved to the lowest speed while the other will get the highest speed) or not. You can use a figure in your justification.

   The new model of AMIAD will be unfair and will starve some clients and make others use more than their share of bandwidth. See the figures below for a comparison



Because the MIAD model rewards the connection already controlling the highest percent of bandwidth, the fastest connection is greedy and consumes all the bandwidth

5. Consider that amazon.com uses a session cookie to track the shopping cart on Susan's browser. Can ebay.com reuse the same cookie? Can amazon.com reuse the same cookie on Bob's browser?

No, the browser maintains cookies linked to an individual site so a cookie set by Amazon wouldn't be sent to eBay. In addition amazon.com sets a unique cookie for each client to persist data like shopping carts, and therefore must have a different cookie for Bob and Susan

6. For the Go-Back-N protocol with the sender window size of 16, what is the minimum number of bits needed for the sequence number field?

With GBN, the max sequence number must be max window size + 1, so max seq # = 16+1=17. $\log_2 17 = 4.\sim$ $\lceil \log_2 17 \rceil = 5$, so the sequence number field must be at least 5

Problem 3 (8 points): Joe is writing programs with a client and a server using stream sockets. The following is the SERVER code that Joe wrote. Can you help Joe to find four errors (there can be more) 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>  /* assume all headers are included correctly */
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_fd; /* listen on server_fd, new connection on new_fd */
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) > 0) { // should be < 0 because negative socket fd indicates error
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    address.sin_family = PF_INET; // should be AF_INET
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = ntohs(PORT); // converts incorrectly from network to host, should be host to network (htons(PORT))

    if (bind(server_fd, (struct sockaddr *)&address, addrlen) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (accept(server_fd, 3) < 0) {
        exit(EXIT_FAILURE);
    }
    if ((new_fd = listen(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
        exit(EXIT_FAILURE);
    }
    int valread = read(server_fd, buffer, 1024);
    printf("%s\n", buffer);
    sendto(new_fd, hello, strlen(hello), 0);  ——> // needs 2 more parameters to specify destination, see corrected version below
    printf("Hello message sent\n");
}
```

sendto(new_fd, hello, strlen(hello), 0, (struct sockaddr *) &address, addrlen)

Problem 4 (12 points): Delay in Application Layer Protocols One end host A visits store.google.com and www.amazon.com sequentially to compare the price of a smart watch.

- (4 points) (a) Assume Host A's local DNS server's cache is empty initially. Therefore, Host A needs to get the IP address of store.google.com via DNS query. Also assume that:
  - Only iterative DNS queries are used.
  - The RTT between Host A and the local DNS server is 20 ms.
  - The RTT between the local DNS server to any authoritative DNS server is 100 ms.
  - Ignore any DNS server's processing time.
  - TTL value for any record is 1 hour.
  - Any domain under google.com is hosted by ns.google.com; any domain under amazon.com is hosted by ns.amazon.com.

  How many milliseconds would have elapsed when Host A gets the IP of store.google.com?

  First one hour Host A → local DNS (20 ms)
  then local DNS must contact the .com server (100 ms) to resolve google.com
  then the local DNS server contacts google.com's name server (100 ms) to resolve store.google.com
  before returning to the Host A with the IP of store.google.com, giving a
  total time of 220 ms

- (4 points) (b) Host A next visits www.amazon.com to compare the price of the same product. Assume www.amazon.com points to a 1000-Byte HTML file which references 4 images. Each referenced image size is 500 Bytes. The *one-way* propagation delay between Host A and Amazon's Web server is 100 ms and the transmission rate is 1 Mbps. When calculating the transmission delay, assume only HTTP response payload counts for that. If Host A uses HTTP 1.0, how many milliseconds would have elapsed when Host A receives all referenced images of www.amazon.com since it inputs the URL www.amazon.com in a browser's address bar?

  To DNS resolve the site www.amazon.com takes 120 ms (20 ms to local DNS server, 100 ms to amazon.com to resolve www.amazon.com).
  Propagation delay for each item received is 200 ms RTT, times 2 for the connection and req (TCP)
  The HTML page takes $\frac{8000}{1000000}$ = 8 ms to transmit
  Each image (4 total) takes 4 200 ms RTT + $\frac{4000}{1000000}$: 4 ms to transmit
  so total time from hitting enter to full page load = 120 + 408 + 4(404) =
  528 + 1616 = 2144 ms

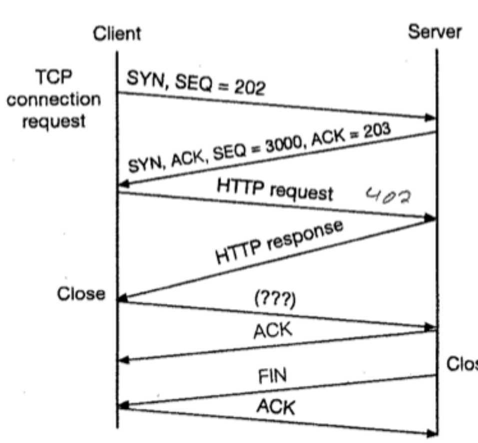
- (4 points) (c) Repeat (b), assuming Host A uses HTTP 1.0 with maximum 4 parallel connections. All other assumptions are the same.

  Parallelizing the connection would still require the same time for DNS resolution, and the first HTML request (120 and 408 ms), but all 4 images could be simultaneously requested, therefore each only taking 404 ms, giving a total time of 120 + 408 + 404 =

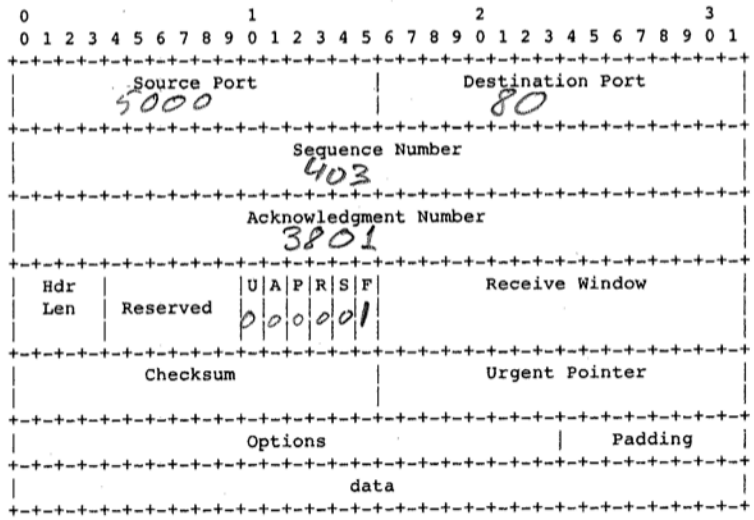
  932 ms

Problem 5 (12 points): TCP Protocol You are asked to fill in as many fields as possible in the TCP header (given in the right subfigure) for the fifth TCP message (marked with (???)) shown in the left figure. The left figure shows the sequence of messages being exchanged between the client and the Web server, starting from TCP connection establishment, HTTP request/response, to TCP connection close.

Write your answers *in decimal format* directly in the right figure. Assume that non-persistent HTTP is used. The HTTP request is 200 Bytes, while the HTTP response from the server is 800 Bytes. Client and Server's IP addresses are `111.111.111.111` and `222.222.222.222`, respectively, and the client uses port 5000 for the TCP connection (hint: the TCP port number used on the Web server is 80).



(a) TCP Message Exchanges



(b) TCP Header

Problem 6 (12 points): Reliable Transfer Protocol In the following scenario, discuss how each reliable transfer protocol reacts:

- (3 points) Will premature timeout (i.e., the retransmission timer has a value smaller than RTT) affect the correctness of reliable transfer protocol? Briefly justify your answer.

  No, it will just cause unnecessary retransmission of the supposedly lost packet. Once the ACK comes back, the window will advance, despite the duplicate packet sent

- (3 points) In the Stop-and-Wait protocol, how does the receiver reacts if a duplicate data packet is received?

  With stop-and-wait, when a duplicate packet is received, the receiver will drop the packet and send an ACK for the duplicate to the sender

- (3 points) Assume cumulative acknowledgment is used and each data segment size is 100B. In the Selective Repeat protocol, how does the sender react when receiving an ACK segment (with acknowledgment number 900B) right after receiving a duplicate ACK segment (with acknowledgment number 800B)?

Receiving a new, higher sequence ACK permits the sender to advance the sender window and begin transmitting more packets, as the cumulative nature of the ACK means each number also ACKs all packets less than that number

- (3 points) In the Go-back-N protocol, if timeout is triggered at the sender before the sender receives the third duplicate ACK, how does the sender react? Justify your answer.

If a timeout occurs in GBN, the sender will retransmit all packets in the current sender window and then await ACKs for each

Problem 7 (20 points): TCP Congestion Control You will work on two scenarios on TCP congestion control. Note that Part 1 and Part 2 are considering different TCP connections; they are not correlated.

1. (8 points) Consider a scenario that a timeout event has been observed at the TCP sender. When the timeout occurs, the congestion window size *cwnd* at the sender is 9 segments. In this scenario, we assume that the receiver's advertised window size is always larger than 10 segments.

    - (2 points) How does TCP congestion control update its *cwnd* upon timeout?

      On timeout, TCP assumes heavy network congestion and therefore drops cwnd to 1

    - (3 points) How does TCP congestion control update its *ssthresh* (i.e., slow start threshold value) upon timeout? Show your steps.

      On timeout, ssthresh = max(MSS·2, cwnd/2). In this case, since cwnd = 9, ssthresh is set to 4

    - (3 points) Can the TCP sender transmit a new segment in addition to retransmitting the segment that experiences timeout? Briefly justify your answer.

      No, because the window decreases to size 1, there can only be one packet in-flight, and that is the retransmitted packet

2. (12 points) Consider another new TCP connection. Assume that all 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* = *cwnd*, use the slow start algorithm. You must draw a diagram to show the intermediate steps on Page 9 to receive full credit.

- TCP uses reliable transfer. The used ACK for each segment is based on cumulative ACK and the acknowledgment number indicates the next expected segment. The receiver acknowledges every segment, and the sender always has data available for transmission.
- Initially *ssthresh* at the sender is set to 8, and *cwnd* as 1. Assume *cwnd* and *ssthresh* are counted in segments, and the transmission time for each segment is negligible (equivalently, you can assume that each segment size is conceptually one unit). Retransmission timeout (RTO) is initially set to 500ms and remains unchanged during the connection lifetime. The RTT is 100ms for all transmissions.
- The connection starts from the initial sequence number of 1 at t=0. Segments with sequence number 4 and 5 arrived at the receiver out of order (i.e., segment 5 arrives before segment 4). Segment with sequence number 7 is lost once. No other misbehavior is observed.

(a) (3 points) The receiver sends an ACK upon receiving segment with sequence number 5. What algorithm should the sender use when receiving this ACK? What is the updated value for *cwnd* and *ssthresh* upon this?

The ~~sender~~ receiver duplicate ACKs packet 3, so ssthresh and cwnd remain unchanged at 8 and 4 respectively

(b) (3 points) The receiver sends an ACK upon receiving segment with sequence number 4. What algorithm should the sender use when receiving this ACK? What is the updated value for *cwnd* and *ssthresh* upon this?

upon getting packet 4, the receiver ACKs packet 5. Therefore the sender continues slow start and increases the cwnd by 2,
giving ssthresh = 8
cwnd = 6

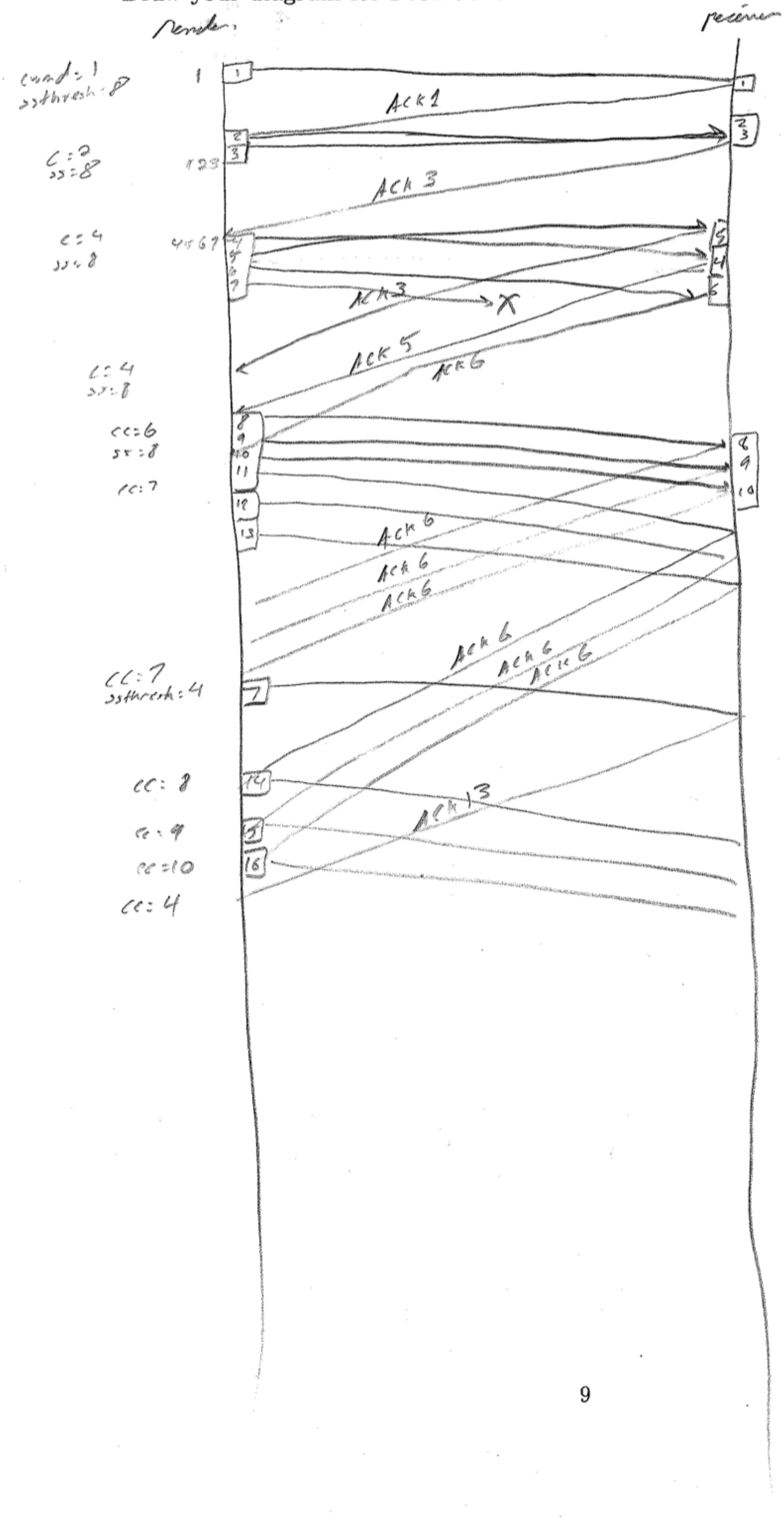
(c) (3 points) The receiver sends an ACK upon receiving the segment with sequence number 10. What is the sequence of actions the sender takes upon receiving this ACK?

the ACK for packet 10 is the 3rd duplicate ACK for packet 6, so the sender goes into fast retransmit and fast recovery, retransmitting packet 7, setting ssthresh = $\lceil 7/2 \rceil$ = 4, and cwnd = ssthresh + 3, or cwnd = 7

(d) (3 points) What is the congestion window size *cwnd* at the sender when the sender transmits segment with sequence number 16?

When the sender transmits packet 16, it is still in the fast recovery mode, so cwnd = 10 (since packet 7 hasn't been ACKed yet) and ssthresh = 4

Draw your diagram for Problem 7.2 here.



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 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 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 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)*;
- Convert multi-byte integer types from host byte order to network byte order:
  - *uint32_t htonl(uint32_t hostlong)*: host to network short
  - *uint16_t htons(uint16_t hostshort)*: host to network long
  - *uint32_t ntohl(uint32_t netlong)*: network to host short
  - *uint16_t ntohs(uint16_t netshort)*: network to host long