1. TCP versus UDP
   a. Describe why an application developer might choose to run an application over UDP rather than TCP

      The answer can be either because of the simplicity or because of the application need. TCP gives congestion control can change sending rate during congestion. An application developer might choose to run an application over UDP rather than TCP because they simply want to avoid TCP congestion control or the application does not require congestion control.

   b. Why is it that voice and video traffic is often sent over TCP rather than UDP in today's Internet? (Hint: The answer we are looking for has nothing to do with TCP's congestion-control mechanism.)

      Because for most firewalls, UDP's traffics can be blocked, for which TCP's traffics cannot in today's Internet.

   c. Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?

      Yes, it is possible. The application runs over UDP can have the data via an application layer protocol.

2. UDP and TCP use 1s complement for their checksums. (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums .)
   a. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? Show all work.

      01010011+01100110=10111001
      10111001+01110100=00101110 <- flip over from highest to lowest

      flip digit of 00101110, we get 1s complement, 11010001

   b. Why is it that UDP takes the 1s complement of the sum to detect errors at the receiver; that is, why not just use the sum?

      UDP has a 1s complement of the sum, so it add all four digits to see the sum contains a zero, if there is a zero, there is an error.

   c. Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

      If there is a 1-bit error, it can be detected, however, if there is a 2-bit error, it cannot be detected.

d. Suppose you have the following 2 bytes: 01011100 and 01100101. What is the 1s complement of the sum of these 2 bytes?

   01011100+01100101=11000001
   flip digit of 11000001, we get 1s complement, 00111110

e. Suppose you have the following 2 bytes: 11011010 and 01100101. What is the 1s complement of the sum of these 2 bytes?

   11011010+01100101=01000000 <- flip over from highest to lowest
   flip digit of 01000000, we get 1s complement, 10111111

f. For the bytes in part (e), give an example where one bit is flipped in each of the 2 bytes and yet the 1s complement doesn't change

   11011011+01100100=01000000 <- flip over from highest to lowest
   flip digit of 01000000, we get 1s complement, 10111111 = 1s complement in e.

3. Host A and Host B are communicating over a TCP connection, and Host B has already received from Host A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.
   a. In the second segment sent from Host A to Host B, what are the sequence number, source port number, and destination port number?

      sequence number = 127 + 80 = 207
      source port number = 302
      destination port number = 80

   b. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?

      sequence number = 127 + 80 = 207
      source port number = 80
      destination port number = 302

   c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?

      acknowledgment number = 127

4. SYN cookies (Section 3.5.6 from textbook or Section 3.6 of RFC 4987).
   a. Why is it necessary for the server to use a special initial sequence number in the SYN-ACK?

      It is necessary because a special initial sequence number contains information such as hash value of IP address, port number and secret number which are essential in order to establish valid communication according to the TCP segment structure.

   b. Suppose an attacker knows that a target host uses SYN cookies. Can the attacker create half-open or fully open connections by simply sending an ACK packet to the target? Why or why not?

      No, the attacker cannot create half-open or fully open connections by simply sending an ACK packet to the target. A server using SYN cookies cannot save connection variables and buffer before a fully open connections established, thus a half-open connection is not possible. Furthermore, in order to establish a fully open connection, it is necessary for the server to use a special initial sequence number in which secret number is required, thus an attacker knows that a target host uses SYN cookies cannot create fully open connections by simply sending an ACK packet to the target.

   c. Suppose an attacker collects a large amount of initial sequence numbers sent by the server. Can the attacker cause the server to create many fully open connections by sending ACKs with those initial sequence numbers? Why or why not?

      No, the attacker cannot cause the server to create many fully open connections by sending ACKs with those initial sequence numbers because the server generate corresponding time stamp in the initial sequence number. The server will not create many fully open connections the expired initial sequence numbers as they are no longer valid.