Problem 1

Suppose that a UDP receiver receives a packet. It then computes the Internet checksum for the received UDP segment, and finds that the result matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred? Explain.

The receiver cannot be absolutely certain that no bit errors have occurred. This is because there may be two bits that flip but still add up to the correct checksum. Thus, the packets may be corrupted but the checksum finds no errors.

Problem 2

Answer the following question about UDP

(a) Consider an application that runs over UDP. Can it still achieve reliable data transfer? If so, how?

(b) Present two application scenarios where UDP is preferred over TCP. Explain why.

(c) Suppose a process in Host A uses a UDP socket with port number 53412. Two other hosts X and Y each send a UDP segment to Host A. Both segments specify destination port number 53412. At host A, will both segments be forwarded to the same socket? If so, can the process at Host A know that these two segments are from two different hosts, and how? If not, would that cause any problem for the process? Discuss and explain. (Hint: think about how you receive a packet over UDP in socket programming)

(a) Yes, it can still achieve reliable data transfer; the reliable data transfer mechanisms must be implemented in the application layer. This way, even though UDP itself is unreliable, the application has mechanisms to ensure reliable data transfer.

(b) *(i)* Live video streaming typically prefers UDP over TCP. The reason for this is because it is faster since it doesn't require a handshake like TCP does. Additionally, in a live streaming scenario, it is more important to deliver the packets that are being sent in real-time rather than waiting for old packets to finish arriving.

(ii) DNS services use UDP over TCP due to the minimal latency of UDP. Moreover, there is less overhead being sent between servers. TCP would be too costly and unnecessary for DNS lookups.

(c) Both segments will be forwarded to the same socket over UDP. The process at Host A will correctly send each segment to its respective destinations because UDP is a 2-tuple of (Source IP Address, Port Number). So, even if two segments have the same port number, if they have different IP addresses, the process at Host A will send each segment to its respective destination.

Problem 3

Consider a reliable data transfer protocol that uses only negative acknowledgments (NAK).

(a) Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to an ACK-based protocol in terms of efficiency? Why?

(b) Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this case, would a NAK-only protocol be preferable to an ACK-based protocol in terms of efficiency? Why?

(a) Using a NAK-only protocol over an ACK-based protocol would not be beneficial if the sender transmits data only infrequently. It would be better to ensure that all segments have been received if the segments are only being sent infrequently since there would be minimal impact on performance.

(b) If the sender has a lot of data to send and the end-to-end connection experiences minimal losses, then a NAK-only protocol is preferable since the sender can send without waiting for an ACK, increasing throughput.

Problem 4

Consider the Go-Back-N protocol with a sender window size of 4 and a sequence number range of 0∼1023 (for simplicity, you do not need to consider sequence number wrap-around). Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that the medium does not reorder messages. Answer the following questions:

(a) List all possible sequence-number-sets (e.g. [k, k+1, k+2, k+3] could be a possible set) inside the sender's window at time t? Justify your answer.

(b) List all possible ACK numbers propagating back to the sender at time t? Justify your answer.

(a) There are 5 cases

$[k-4, k-3, k-2, k-1]$	No ACK for k - 4
$[k-3, k-2, k-1, k]$	ACK for k - 4
$[k-2, k-1, k, k+1]$	ACK for k - 3
$[k-1, k, k+1, k+2]$	ACK for k - 2
$[k, k+1, k+2, k+3]$	ACK for k - 1

(b) At time t, if the sender hasn't sent packet k yet (worst case), the possible ACK numbers propogating back to the sender are k - 4, k - 3, k - 2, k - 1. Since we use cumulative acknowledgement, we won't get an ACK earlier than k - 4.

Problem 5

Answer True or False to the following questions and briefly justify your answer:

(a) With the Selective Repeat protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its sender's window.

(b) With Go-Back-N, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

(c) The Stop&Wait protocol is the same as the Selective Repeat protocol with a sender and receiver window size of 1.

(d) Selective Repeat can buffer out-of-order-delivered packets, while Go-Back-N cannot. Therefore, Selective Repeat saves network communication cost (by transmitting less) at the cost of additional memory.

(a) True. Consider a long delay. If a receiver sends an ACK and it gets lost in the network, the sender may time out and resend a packet, moving the packet in question outside of the window. The sender, upon receiving the ACK, discards it as a dupliate.

(b) True. Since Go-Back-N uses cumulative acknowledgement, there may be a scenario where ACK_1 may come after ACK_2. But by that time, packet 1 would have fallen out the window.

(c) True. Stop&Wait waits for each individual packet. That is the same as saying we have a window size of one.

(d) True. Go-Back-N uses cumulative acknowledgement, so it must be in-order by definition. Selective Repeat is able to buffer out-of-order packets, which implies that we may need a memory buffer to store these packets. Therefore, it can send only necessary packets, reducing networking cost.