Solutions to the Sample Questions on Transport Layer -1

Q1) Which protocol – Go-Back-N or Selective-Repeat - makes more efficient use of network bandwidth? Why?

A1. Selective repeat makes more efficient use of network bandwidth since it only retransmits those messages lost at the receiver (or prematurely timed out). In Go-Back- N, the sender retransmits the first lost (or prematurely timed out) message as well as all following messages (without regard to whether or not they have been received).

Q2) Consider a reliable data transfer protocol that uses only negative acknowledgements. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

A2. In a NAK only protocol, the loss of packet x is only detected by the receiver when packet x+1 is received. That is, the receiver receives x-1 and then x+1, only when x+1 is received does the receiver realizes that x was missed. If there is a long delay between the transmission of x and the transmission of x+1, then it will be a long time until x can be recovered, under a NAK only protocol.

On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent – a significant reduction in feedback in the NAK-only case over the ACK-only case.

Q3) If the RTT from London to Cape Sydney is 120ms and all links in the network have a 155 Mbits/second data-rate, how much data can fit in the "pipe"? Hint: Bandwidth Delay Product. Express your answer in bytes.

A3. Bandwidth Delay product BDP = 155 Mbits/sec * 120 ms = 18.6 x 10^6 bits = 2,325,000 bytes will fit in the pipe.

Q4) Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?

A4. Only if the application itself implements reliability measures, such as ACK, retransmission, timer, etc.

Q5) Consider a TCP connection between Host A and Host B. Suppose that the TCP segments travelling from Host A to Host B have source port number x and destination port number y. What are the source and destination port numbers for the segments travelling from Host B to Host A?

The port numbers are swapped. They would become source port: y and destination port: x for segments travelling from Host B to Host A.

Q6) Suppose that the UDP receiver computes the Internet checksum for the received UDP segment and finds that it matches the value carried in the checksum field. Can the receiver be absolutely sure that no bit errors have occurred? Explain. Would things be different with TCP?

A6. No, the receiver cannot be absolutely certain that no bit errors have occurred. This is because of the manner in which the checksum for the packet is calculated. If the corresponding bits (that would be added together) of two 16-bit words in the packet were 0 and 1 then even if these get flipped to 1 and 0 respectively, the sum still remains the same. Hence, the 1s complement of the sum the receiver calculates will also be the same. This means the checksum will verify even if there was transmission error. Since TCP uses the same 16 bit Internet checksum mechanism, the above would hold true with TCP as well.

Q7) In protocol rdt3.0, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence numbers of the packet they are acknowledging). Why is that the ACK packets do not require sequence numbers?

A7. To best answer this question, consider why we needed sequence numbers in the first place. We saw that the sender needs sequence numbers so that the receiver can tell if a data packet is a duplicate of an already received data packet. In the case of ACKs, the sender does not need this info (i.e., a sequence number on an ACK) to detect a duplicate ACK. A duplicate ACK is obvious to the rdt3.0 sender, since when it has received the original ACK it transitioned to the next state. The duplicate ACK is not the ACK that the sender needs and hence is ignored by the rdt3.0 sender.