Home Page: http://www.public.asu.edu/~bhao2

Homework 3

Problems:

2, 7, 14, 15, 20 and 25

Question: 2

Consider Figure 3.5. What are the source and destination port values in the segments flowing from the server back to the clients' processes? What are the IP addresses in the network-layer datagrams carrying the transport-layer segments?

Answer:

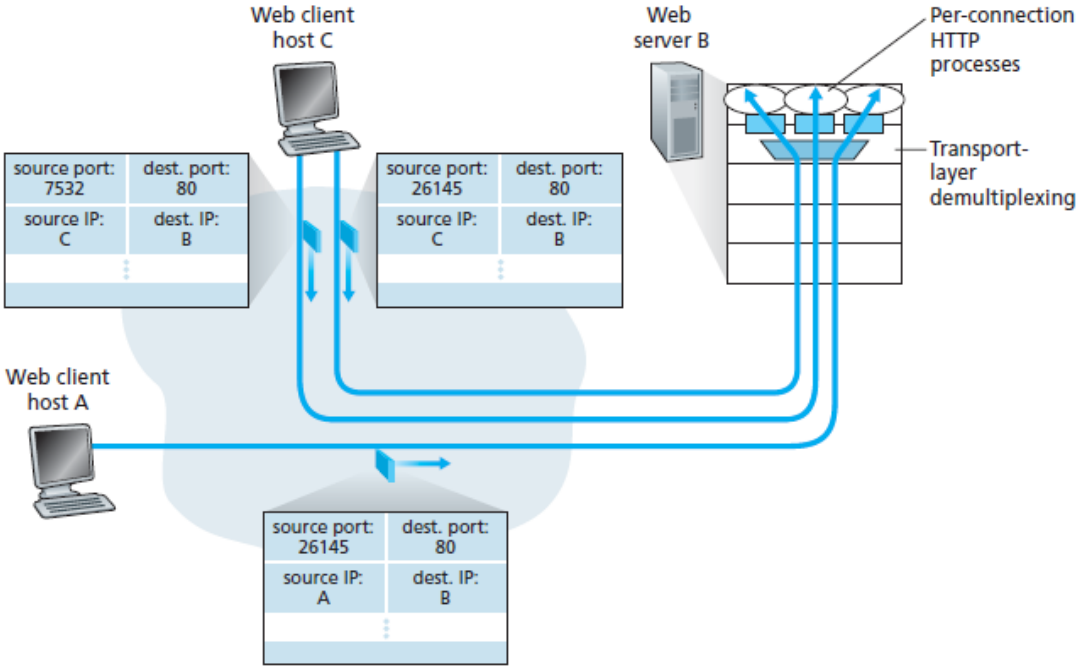


Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

The source port and destination port will swap for the server back to the client' processes. For the Host A, it is sending segment with source port 26145 and destination port 80. Therefore the server B will send the response back to Host A using source port 80 and destination port 26145. For the Host C , server B will send the response back with source port 80 and destination port 26145 and 7532.

The IP addresses in the datagrams that are heading back to A or C will have the source IP set to the servers IP address, and the destination IP set to the appropriate clients address, which it specified in its own request it had originally sent to the server.

Question: 7

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 number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

Answer:

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 tell detect a duplicate ACK. A duplicate ACK is obvious to the rdt3.0 receiver, 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.

Question: 14

Consider a reliable data transfer protocol that uses only negative acknowledgments. 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?

Answer:

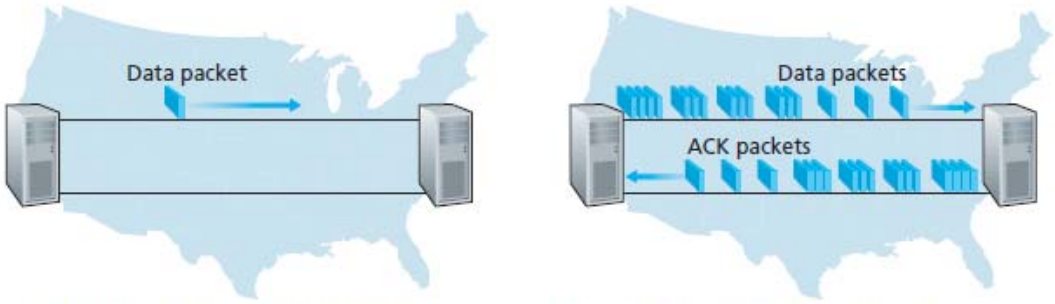
In a NAK only protocol, the loss of packet x is only detected by the receiver when packet x+1 is received. That is, the receivers receives x-1 and then x+1, only when x+1 is received does the receiver realize that x was missed. If there is a long delay between the transmission of x and the transmission 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.

Question: 15

Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data.

Answer:



a. A stop-and-wait protocol in operation b. A pipelined protocol in operation

Figure 3.17 ♦ Stop-and-wait versus pipelined protocol

Reference: book page 217

Since $d_{trans} = \frac{L}{R}$ and we know the size of a packet is 1500bytes, the link is 1Gbps, thus

Therefore, $d_{trans} = \frac{1500bytes * 8}{10^9} = 0.00001.2\ seconds = 0.012\ milliseconds$, this means a packet needs 0.012 ms to be sent on the link.

According to the problem, the utilization = 0.98 is required.

Since $U_{\text{sender}} = \frac{L/R}{RTT + L/R}$, and we know RTT = 30, $d_{trans} = \frac{L}{R}$ = 0.012 ms

Let the window size be x:

$$U = \frac{\frac{L}{R} * \text{x}}{RTT + \frac{L}{R}} = \frac{0.012 * x}{30 + 0.012} = \frac{0.012 * x}{30.012} = 0.98$$

So, x = 2450.98

The window size would have to be approximately 2451 packets.

Question: 20

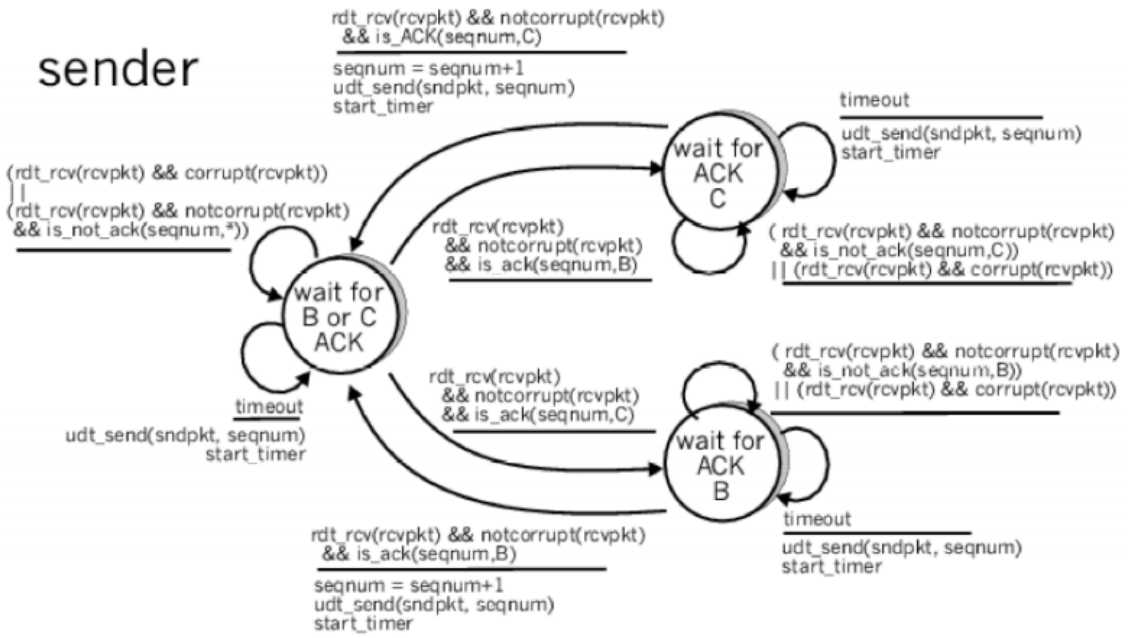
Consider a scenario in which Host A and Host B want to send messages to Host C. Hosts A and C are connected by a channel that can lose and corrupt (but not reorder) messages. Hosts B and C are connected by another channel (independent of the channel connecting A and C) with the

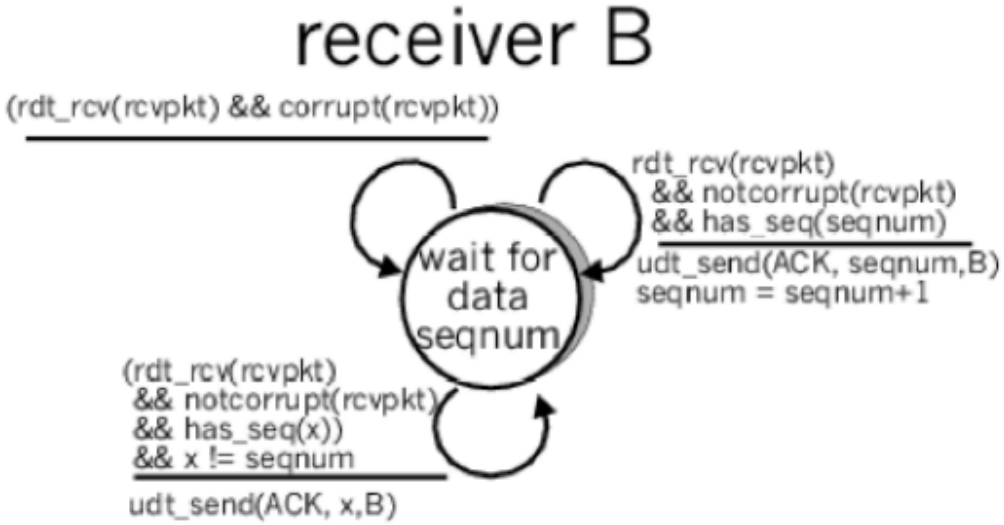
same properties. The transport layer at Host C should alternate in delivering messages from A and B to the layer above (that is, it should first deliver the data from a packet from A, then the data from a packet from B, and so on). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A and B to C, with alternating delivery at C as described above. Give FSM descriptions of A and C. (Hint: The FSM for B should be essentially the same as for A.) Also, give a description of the packet format(s) used.

Answer:

This problem is a variation on the simple stop-and-wait protocol (rdt3.0). Because the channel may lose message and because the sender may resend a message that one of the receivers has already received (either because of a premature timeout or because the other receiver has yet to receive the data correctly), sequence numbers are needed. As in rdt3.0, a 0-bit sequence number will suffice here.

In this problem, the sender state indicates whether the sender has received an ACK from B (only), from C (only) or from neither C nor B. The receiver state indicates which sequence number the receiver is waiting for.





Question: 25

We have said that an application may choose UDP for a transport protocol because UDP offers finer application control (than TCP) of what data is sent in a segment and when.

a. Why does an application have more control of what data is sent in a segment?

b. Why does an application have more control on when the segment is sent?

Answer:

a. Consider sending an application message over a transport protocol. With TCP, the application writes data to the connection's send buffer' and TCP will grab bytes without necessarily putting a single message in the TCP segment (TCP may put more or less than a single message in a segment). UDP, on the other hand, encapsulates in a segment whatever the application gives it. So, if the application gives UDP an application message, this message will be payload of the UDP segment. Thus, with UDP, an application has more control of what data is sent in a segment.
b. With TCP, due to flow control and congestion control, there may be significant delay from the time when an application writes data to its send buffer until when the data is given to the network layer. UDP does not have delays due to flow control congestion control.