Chapter 3 Assignments

P13. Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data 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.

P23. 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 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 singe message in a segment. UDP, on the other hand, encapsulates in a segment whatever the application gives it; so that, if the application gives UDP an application message, this message will be the 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 and congestion control.

P34. Compare GBN, SR, and TCP (no delayed ACK). Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost in the channel) by the receiving host (Host B) and the sending host (Host A) respectively. Suppose Host A sends 5 data segments to Host B, and the 2nd segment (sent from A) is lost. In the end, all 5 data segments have been correctly received by Host B.

a. How many segments has Host A sent in total and how many ACKs has Host B sent in total? What are their sequence numbers? Answer this question for all three protocols.

b. If the timeout values for all three protocol are much longer than 5 RTT, then which protocol success fully delivers all five data segments in shortest time interval?

Answer:

a).

GoBackN:

A sends 9 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later re-sent segments 2, 3, 4, and 5.

B sends 8 ACKs. They are 4 ACKS with sequence number 1, and 4 ACKS with sequence numbers 2, 3, 4, and 5.

Selective Repeat:

A sends 6 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later re-sent segments 2.

B sends 5 ACKs. They are 4 ACKS with sequence number 1, 3, 4, 5. And there is one ACK with sequence number 2.

TCP:

A sends 6 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later re-sent segments 2.

B sends 5 ACKs. They are 4 ACKS with sequence number 2. There is one ACK with sequence numbers 6. Note that TCP always send an ACK with expected sequence number.

b). TCP. This is because TCP uses fast retransmit without waiting until time out.

P37 Consider Figure 3.58. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.

a. Identify the intervals of time when TCP slow start is operating.

b. Identify the intervals of time when TCP congestion avoidance is operating.

c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

e. What is the initial value of ssthresh at the first transmission round?

f. What is the value of ssthresh at the 18th transmission round?

g. What is the value of ssthresh at the 24th transmission round?

h. During what transmission round is the 70th segment sent?

i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?

j. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size at the 19th round?

k. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?

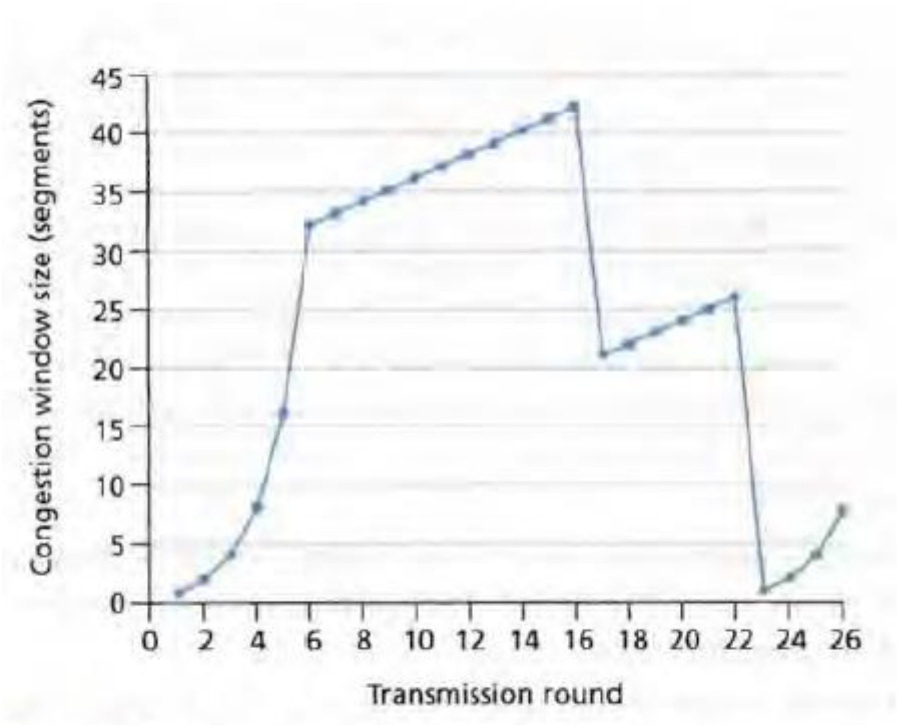

Figure 3.58 • TCP window size as a function of time

Answer:

a) TCP slow start is operating in the intervals [1,6] and [23,26]

b) TCP congestion avoidance is operating in the intervals [6,16] and [17,22]

c) After the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.

d) After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.

e) The threshold is initially 32, since it is at this window size that slow start stops and congestion avoidance begins.

f) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 16, the congestion windows size is 42. Hence the threshold is 21 during the 18th transmission round.

g) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 22, the congestion windows size is 26. Hence the threshold is 13 during the 24th transmission round.

h) During the 1st transmission round, packet 1 is sent; packet 2-3 are sent in the 2nd transmission round; packets 4-7 are sent in the 3rd transmission round; packets 8-15 are sent in the 4th transmission round; packets 16-31 are sent in the 5th transmission round; packets 32-63 are sent in the 6th transmission round; packets 64 – 96 are sent in the 7th transmission round. Thus packet 70 is sent in the 7th transmission round.

i) The congestion window and threshold will be set to half the current value of the congestion window (8) when the loss occurred. Thus the new values of the threshold and window will be 4.

j) Threshold is 21, and congestion window size of 17th round is reset 1 then do slow start, so the congestion window size of 19th round is 4.

k) round 17, 1 packet; round 18, 2 packets; round 19, 4 packets; round 20, 8 packets; round 21, 16 packets; **round 22, 21 packets** (not 32 packets!). So, the total number is 52.

P40. Host A is sending an enormous file to Host B over a TCP connection. Over this connection there is never any packet loss and the timers never expire. Denote the transmission rate of the link connecting Host A to the Internet by *R* bps. Suppose that the process in Host A is capable of sending data into its TCP socket at a rate S bps, where S = 10 *R*. Further suppose that the TCP receive buffer is large enough to hold the entire file, and the send buffer can hold only one percent of the file. What would prevent the process in Host A from continuously passing data to its TCP socket at rate S bps? TCP flow control? TCP congestion control? Or something else? Elaborate.

Answer:

In this problem, there is no danger in overflowing the receiver since the receiver's receive buffer can hold the entire file. Also, because there is no loss and acknowledgements are returned before timers expire, TCP congestion control does not throttle the sender. However, the process in host A will not continuously pass data to the socket because the send buffer will quickly fill up. Once the send buffer becomes full, the process will pass data at an average rate or R << S.

D3 (don't submit) Read the research literature to learn what is meant by *TCP friendly.* Also read the Sally Floyd interview at the end of this chapter. Write a one-page description of TCP friendliness.

Answer:

TCP-Friendly Rate Control (TFRC) is a congestion control mechanism designed for unicast flows operating in an Internet environment and competing with TCP traffic. The goal is to compete fairly with TCP traffic on medium timescales, but to be much less variable than TCP on short timescales.

TCP congestion control works by maintaining a window of packets that have not yet been acknowledged. This window is increased by one packet every round trip time if no packets have been lost, and is decreased by half if a packet loss is detected. Thus TCP's window (and hence throughput) is a function of the losses observed in the network and the round trip time experienced by the flow. At fixed time intervals, the sender computes the loss rate observed during the previous interval an updates the sending rate using the TCP response function. Since the protocol adjusts its send rate only at fixed time intervals, the transient response of the protocol is poor at lower time scales. In addition, computing the loss rate at fixed time intervals make the protocol vulnerable to changes in RTT and sending rate. Compares the performance of TFRC and TFRCP, and finds that TFRC gives better performance over a wide range of time scales.

The idea behind TFRC is to measure the loss probability and round trip time and to use these as the parameters to a model of TCP throughput. The expected throughput from this model is then used to directly drive the transmit rate of a TFRC flow.

In addition, computing the loss rate at fixed time intervals make the protocol vulnerable to changes in RTT and sending rate. Compares the performance of TFRC and TFRCP, and finds that TFRC gives better performance over a wide range of time scales.