Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.02 **Solutions to Chapter 19** Updated: May 8, 2012

---

1. (Not surprisingly,) Louis is wrong. Imagine that the sender sends packet k and then retransmits k. However, the original transmission and the retransmission get through to the receiver. The receiver sends an ACK for k when it gets the original transmission, and in response the sender sends packet $k+1$. Now, when the sender gets an ACK, it cannot tell whether the ACK was for packet k (the retransmission), or for packet $k+1$!

2. Because T_p is independent of packet size, and smaller packets have a lower transmission time over the link, what matters for this question is the processing time for the smallest packet. The maximum throughput of the stop-and-wait protocol is 1 packet every round-trip time (RTT), which in our case is the sum of the transmission time and T_p. The transmission time for a 540 bit packet at 54 Megabits/s is 10 microseconds. Hence, if T_p^* is the maximum allowable processing time, we have:

$$540 \text{ bits}/(10 \text{ microseconds} + T_p^*) = 27 \text{ Megabits/s},$$

   giving us $T_p^* = 10$ microseconds.

3. See PSet.

4. α should decrease. The smaller the value of α, the more history in the EWMA, and the filter behaves as a better low-pass filter (see Figure 19-4 in Chapter 19). One can write out a mathematical expression, but it isn't necessary to answer the question.

5. The important point that we wanted to get across from this question is that due to the smoothing of the EWMA filter in the RTT estimation, if there is a sudden jump in actual RTT, the RTT estimate may take quite a while to reflect the new RTT. That, in turn, can cause spurious retransmissions. Using the mean deviation in the RTO calculation helps avoid spurious transmissions.

   After the actual RTT becomes R, we will avoid spurious retransmissions when the RTT estimate becomes R/β. Suppose this situation happens at the n^{th} sample. Then, $y(n) = R/\beta$. We need to solve for n.

   Now,

$$\begin{aligned}
y(n) &= R\alpha + R\alpha(1-\alpha) + ... + R\alpha(1-\alpha)^{n-1} + r_0(1-\alpha)^n \\
&= r_0(1-\alpha)^n + \alpha R[1 + (1-\alpha) + .. + (1-\alpha)^{n-1}] \\
&= r_0(1-\alpha)^n + \alpha R\frac{1-(1-\alpha)^n}{1-(1-\alpha)} \\
&= r_0(1-\alpha)^n + R[1-(1-\alpha)^n] \\
&= r_0(1-\alpha)^n + R - R(1-\alpha)^n
\end{aligned}$$

   Substituting $y(n) = R/\beta$ and solving for n, we get $n = \log_{(1-\alpha)}(R/(R-r_0))(1-1/\beta)$

Hari Balakrishnan

With $\alpha = 1/8, \beta = 2$ and neglecting r_0, we get

$n = \log_{7/8}(1/2) = 5.19$.

Hence, *after 6 samples*, there are no spurious retransmissions.

6. The largest sequence number that a receiver could have *possibly* received is `last_sent`. The size of the receiver buffer can become as large as `last_sent - first_unacked`. One might think that we need to add 1 to this quantity, but observe that the only reason any packets get added to the buffer is when some packet is lost (i.e., at least one of the packets in the sender's unacked buffer must have been lost).

   We also need to handle the case when all the packets sent by the sender have been acknowledged—clearly, in this case, the sender should be able to send data.

   Hence, if the sender sends a new packet

   ```
   if len(unacked_packets) == 0 or last_sent - first_unacked < W
   ```

   then the desired requirement is satisfied.

7. The answers to the different parts of this question are related to the bandwidth-delay product concept.

   (a) The throughput is 10 packets per RTT, or 100 packets/s, which is 100 Kbytes/s. That's **10%** of the bottleneck rate of 1 Mbyte/s.

   (b) When the window exceeds 256 packets, the sequence number field wraps around. If the first packet in such a window is lost, then the retransmission and the transmission of a later packet may be confused for one another.

   If the window defined as the difference between the last transmitted packet sequence number and the last in-order acknowledgment, then the sender can't send a packet beyond $A + W$, where A is the last in-order acknowledgment and W is the window. The largest possible W is the bandwidth-delay product. As a result, confusion from a wrapped sequence number will arise only when the bandwidth-delay product exceeds 256 packets, which implies that the highest bottleneck link is $256 \cdot 1000/0.1$ bytes/s = 2.56 Megabits/s.

   (c) (i) 9 packets. If the window size is W and no packets are lost, then packets get delivered to the application by the receiver transport protocol and the buffer doesn't exceed 1. On the other hand, suppose ℓ packets are lost in the window. Then, the number of packets buffered at the receiver will be equal to $W - \ell$—one for each packet received. By the definition of the window, no packets with sequence number bigger than $a + W$ will be sent by the sender, where a is the sequence number of the last in-sequence ACK. (I.e., packet number $a + 1$ has not yet reached the receiver.) The largest value of $W - \ell$ is $W - 1$, which in our case is 9 packets.

   (ii) 90 packets. The sender's timeout is 1 second, which is 10 RTT intervals. In each RTT, the sender sends **9** packets: in the first RTT, one of the ten packets is lost, and in each subsequent RTT, the sender transmits one packet for each ACK it receives, so it sends 9 packets in each succeeding RTT. After 10 RTTs, it times out and retransmits the lost packet; when that packet is received, the receiver will have $10 \cdot 9 = 90$ packets in its buffer. (If one ends up also storing the retransmission of the lost packet in the

buffer before delivering all the packets in order to the application, we will need one more packet in the buffer, making the answer 91 packets.)

Note though that in the worst case, if additional losses could occur (which we have asked you to assume away), the maximum buffering required at the receiver could be *unbounded* because that first lost packet could repeatedly get lost!

8. See PSet.

9. See PSet.

10. (a) $W = C \cdot T$. Note that T is the RTT *including queueing*. If T were instead the RTT without queueing, then $W = CT + Q$.

    (b) Clearly, $W = W_{min} + Q$, where W_{min} is the smallest window size that gives 100% utilization. A smaller window than that would keep the network idle some fraction of the time. Hence, $W_{min} = C \cdot T - Q$.

    (c) Packets start getting dropped when the window size is $W + 1$, i.e., when it is equal to $CT + 1$.

11. (a) **Choice (ii).** The RTT, which is the time taken for a single packet to reach the receiver and the ACK to arrive at the sender, is about 21 milliseconds: $10 + 10$ for the propagation delays in each direction and 1 ms for the transmission delay of a data packet (the transmission delay of the ACK is 25 times less and may be ignored). Hence, the throughput is 10 packets / 21 milliseconds = 476 packets per second. If one ignored the transmission time, which is perfectly fine given the set of choices, one would estimate the throughput to be about 500 packets per second.

    (b) 1. **Choices (i) and (ii).** When $W = 10$, the throughput is about 476 packets/s. If we double the window size, throughput would double to 952 packet/s. If we reduce the propagation time of the links, throughput would roughly double as well. The new throughput would still be smaller than the bottleneck capacity of 1000 packets/s.

    2) $W = 50$: **Choice (iii).** When $W = 50$, throughput is already 1000 packets/s. At this stage, doubling the window or halving the RTT does not increase the throughput. If we double the speed of the link between the Switch and Receiver, the bottleneck becomes 2000 packets/s. A window size of 50 packets over an RTT of 20 or 21 milliseconds has a throughput of more than 2000 packet/s. Hence, doubling the bottleneck link speed will double the throughput when $W = 50$ packets.

    3) **None.** When $W = 30$, throughput is already 1000 packets/s. Now, if we double the window or halve the RTT, the throughput won't change. An interesting situation occurs when we double the link speed, because the bottleneck link would now be capable of delivering 2000 packets/s. But our window size is 30 and RTT about 20 milliseconds, giving a throughput of about 1500 packets/s (or if we use 21 milliseconds, we get 1428 packet/s). That's an improvement of about 50%, far from the doubling we wanted. None of the techniques work.

12. (a) The protocol is "at least once".

    (b) See PSet.

    (c) See PSet.

13. (a) The round-trip propagation time is 10 milliseconds. The transmission time for a tweet is (1000 bytes / 100 kbytes/s) = 10 milliseconds. The ACKs have negligible transmission time, so the RTT is $10 + 10 = 20$ milliseconds.

    (b) The expected time for a tweet to get an ACK is equal to

$$\text{RTT } + \frac{1-p}{p} \cdot \text{Timeout},$$

    where p is the probability of a given tweet receiving an ACK and RTT the round-trip time. Plugging in the numbers, we find that the expected time is 50 milliseconds. Hence, the rate at which Sir Tweetsalot can send successful tweets is $1000/50 = 20$ tweets/s. The capacity of the network is 100 kbytes/s, and if that were used at 100% utilization for tweets, we would be able to send 100 tweets/s. Therefore, the utilization is 20%.

    Note: If one *wrongly* sets the RTT to 10 ms, a tempting choice because that is the given round-trip propagation time, then the answer would be 25%. In contrast, if the RTT is incorrectly estimated at more than 90 ms from Part (a), then one may wrongly end up concluding that there are spurious retransmissions!

14. (a) $\frac{S}{R}$. Each data packet has size S bits, and the speed of the link is R bits per second.

    (b) $\frac{nD}{c}$. Total distance to be travelled is nD since each link has length D meters, and there are n such links. The propagation speed is c meters/second.

    (c) $\frac{nS}{R} + \frac{nK}{R} + \frac{2nD}{c} + T_p$. We need to consider the following times:
    - Transmit data across n links: $\frac{nS}{R}$ using result from 12A.
    - Transmit ACK across n links: $\frac{nK}{R}$ also using result from 12A
    - Propagate data across n links and ACKS across n links: $\frac{2nD}{C}$
    - Total time to process the data and the ACK: T_p

    (d) $(1-p)^n$. Probability of loss in a link is p, so probability of no loss in a link is $1-p$. Since link losses are independent, probability of no loss in n links is $(1-p)^n$. No loss in n links means the data gets from A to B successfuly.

15. See PSet.

16. No. For example, see Figure 16.

17. (a) The bandwidth-delay product of the connection is 10 packets (bottleneck rate times the minimum RTT). With a window size of 8, queues will not yet have built up, so the throughput is 80 packets/second.

    (b) Queues will have built up; the bottleneck link is now saturated, so the throughput is 100 packets/second.

    (c) 11 packets. The bandwidth-delay product is 10 packets. It's probably reasonable to accept an answer of 10 packets too.

    (d) 30 packets. When $W = C \times RTT_{min} + Q = 100 \times 0.1 + 20 = 30$, the queue will be always full. When W is greater than 30, there will be queue-overflow and packets will be dropped by the queue.

18. See PSet.

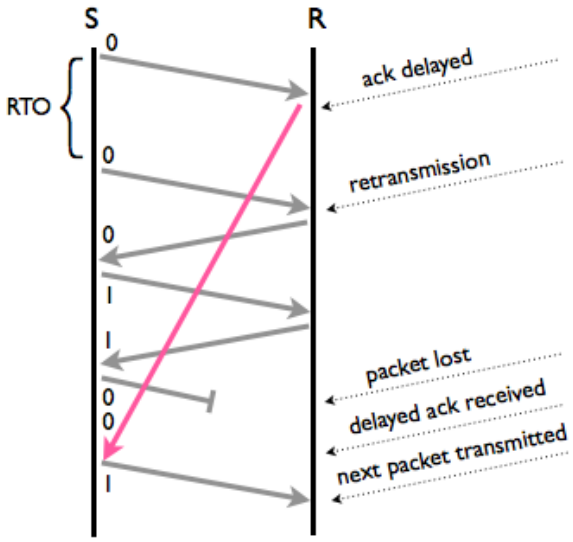

Figure 1: Example for Problem 16.

19. (a) The throughput will **increase** (by a little). Since the channel is completely saturated because the window size is $2 \cdot B \cdot RTT_{min}$ and larger than the bandwidth-delay product, the utilization is $1 - l$. If l decreases, the throughput increases.

    (b) The resulting bandwidth-delay product increases by a factor of 1.8, but is still smaller than the window size, so the throughput will **remain the same**.

    (c) The throughput will **decrease** because the window size will now be strictly smaller than the bandwidth-delay product, and the bottleneck link will no longer be saturated.