HW #2 Solutions

ELEN E6761 - Computer Communication Networks
Fall 2000 Prof. Rubenstein

Please turn your homework into Vassilis Stachtos' mailbox, E2 (oustide of the Electrical Engineering office, fax it by 5pm, or make some other arrangement with Vassilis. On-campus students can fax to Professor Rubenstein at (212) 932-9421 (remember to include your name and that the fax is for Professor Rubenstein). CVN students can fax as above, or fax to the CVN fax number. LATE ASSIGNMENTS WILL NOT BE ACCEPTED!! Note: don't wait until the last minute to fax - only one can go through at a time.

1. Given a linespeed of l bits / second, a packet size of k bits, a round trip time of R, and a window size of W, what is the rate of communication? Assume that there is no packet loss and that R includes the time it takes the receiver to receive, process and acknowledge the data packet, and also includes the time it takes the sender to receive and process the acknowledgment. However, it does not include the time that it takes the sender to place the data packet onto the wire.

   Answer: If there were no window limitation, the rate of communication would be l bits / second, so this is the upper bound on the transmission rate. The window permits kW bits in the network at any given time. The actual RTT for any packet must also include the transmission time of the next data packet, which is k/l, making the total RTT equal to $R + k/l$. Little's Law gives us that the rate of communication in this case is $kW/(R+k/l)$ (a packet is "in the system" until its acknowledgment is complete allowing the sender to insert the next packet. Thus, the rate of communication is $\min(kW/(R + k/l), l)$.

2. Consider a reliable data transfer protocol that transfers data using a window of size w. Assume that packets can be lost, but cannot be reordered. Show that the minimum number of sequence numbers that the protocol must utilize to guarantee error-free transfer when the transfer is performed in the style of

   (a) Selective Repeat is $2w$.

   Answer: When saying the ith packet, we refer to the transmission of the ith portion of a stream (i.e., two packets i and j transmit identical portions of the stream if and only if $i = j$. Let $s(i)$ indicate the sequence number of the ith packet. When we have n distinct sequence numbers, it will be the case that $s(i) = s(j)$ whenever $i \pmod n = j \pmod n$.

   To see that $2w-1$ is an insufficient number of sequence numbers, consider the following scenario. The sender initially sends packets 0 through $w-1$. The receiver receives all these packets and ACKs them all. All the ACKs are lost, so the sender retransmits packet 0. The receiver has rolled its window past $w-1$ so that its window now covers packets w through $2w-1$ so that it expects the $2w-1$st packet, and note that $s(2w-1) = s(0)$. When the sender retransmits packet 0, the receiver incorrectly infers the packet to be the $2w-1$st packet.

   To prove that $2w$ sequence numbers are sufficient, we must show that at any time, if the receiver is able to receive packet i, then it should not be possible for the sender to transmit packet j where $s(j) = s(i)$. Assume WLOG (without loss of generality) that the receiver's window is at $[j, j+w-1]$, i.e., the receiver will accept packets j through $j + w - 1$. This means that the receiver has already received all packets up to and including packet $j-1$ and has not yet received packet j. This also means that the sender must have already sent packet $j-1$, so its window must *at least* be at $[j-w, j-1]$, though it can be as far as $[j, j+w-1]$ (it cannot move any farther until the receiver acknowledges j, which has not yet happened since the receiver's window is only be positioned at $[j, j+w-1]$ when the receiver has not yet received j).

   Also note that since there is no reordering within the network, any packet whose number is less than $j - w$ can no longer be in transit: $j-1$ could only have been transmitted after the bottom of the sender's window was already shifted to $j - w$. Hence, the only packets that can be in transit in the network at this time are $j-w$ through $j+w-1$. This is a sequence of $2w$ packets, so with $2w$ sequence numbers, each packet that could possibly be in transit has a unique sequence number. Thus, any of the packets j through $j+w-1$ that the receiver is possibly waiting to receive has a sequence number that is unique over all packets possibly in transit.

Last, we must verify that the sender does not misinterpret any receiver acknowledgments that are potentially in transit when the receiver's window is located at $[j, j+w-1]$. Let the sender window currently be located at $[j-i, j-i+w-1]$, where $0 \leq i \leq w$ (above we determined that these were the only valid locations for the sender window). This means that the sender will only respond to sequence numbers $s(j-i), s(j-1+1), \cdots, s(j-i+w-1)$, since only these sequence numbers currently reside in its window. This means that the sender has already received the acknolwedgment for packet $j-i-1$, so any ACKs in transit are acknowledging packets received after the receiver had already received packet $j-i-1$. Thus, the lowest numbered acknowledgment that can possibly still be in transit is $j-i-w$. But all packets $j-i-w$ through $j-i-1$ have sequence numbers that differ from $s(j-i), s(j-1+1), \cdots, s(j-i+w-1)$, so the sender will simply ignore any such acknowledgments since they cannot correspond to packets that reside within its current window. Note that no two different packets, drawn from the range of packets with numbers between (and including) $j-i$ through $j+w-1$, can have the same sequence number. This means the sender can uniquely determine the identity of any packet in its window from its sequence number.

(b) Go-Back-N (assume the receiver discards any packets that cause a gap in the sequence numbering) is $w+1$.

Answer: To see that w sequence numbers is not enough, consider the following scenario. The sender initially sends packets 0 through $w-1$. All of these are received and acknowledged by the receiver, but all the acknowledgments are lost, so the sender retransmits packet 0. The receiver has rolled its window to contain packets w through $2w-1$, but packet w has sequence number 0, so when the sender retransmits packet 0, the receiver incorrectly infers the packet is the wth packet.

To prove that $w+1$ sequence numbers are sufficient, we must show (as we did in the case of selective repeat) that at any time, if the receiver is willing to receive packet i, then no packet j should be transmitted where $s(j) = s(i)$. Again WLOG, assuming that the receiver's window is at $[j, j+w-1]$, the same arguments apply so that we know the sender's window is located between $[j-w, j-1]$ and $[j, j+w-1]$, and that only those packets within this range can be in transit within the network at this time. Note that in the Go-Back-N protocol, when the j is the bottom-most packet in the receiver's window, it will only accept a packet with sequence number $s(j)$, and reject any other packets, even if they lie within the receiver's window above j. Note that with $w+1$ sequence numbers, if we choose any other packet i in the range from $j-w$ to $j+w-1$, then $s(i) = s(j)$ only if $i = j$. Hence, the only packet that the receiver will accept at this time is packet j.

Finally, we must prove that the sender never misinterprets any acknowledgments. Again assume that the sender's window is located at $[j-i, j-i+w-1]$ for some $0 \leq i \leq w$. Then the sender has already received an ACK for $j-i$ (recall that the transmission of ACK k indicates that the receiver is awaiting packet k.) Note that in Go-Back-N, the sequence of packet number acknowledgments transmitted by the receiver is a monotonically non-decreasing sequence (each packet number ACK'd is either the same or larger than that of the previous packet ACK'd). Since an ACK for $j-i$ was already received, the ACKs in transit must lie between $j-i$ and j. For the sender to erroneously advance its window, it would have to receive some ACK with sequence number $s(j-i+k)$ for $0 \leq k < w$, where the actual packet number for the ACK was $j-i+k-(w+1) = j-i-w-1+k$. However, for all values of k where $0 \leq k < w$, all such ACKs are no longer in transit. Thus, the sender cannot misinterpret the meaning of an ACK and erroneously roll its window forward.

(c) 10% EXTRA CREDIT: Assume now that packet ordering can be shifted by up to n, i.e., the jth packet transmitted by the sender might arrive after the $j+n-1$st packet. How does this effect the required minimum size of the sequence numbering for both cases above? FORMAL PROOF REQUIRED TO RECEIVE FULL CREDIT!

Answer: The sequence number space needs to be increased by n.

3. Assume the bounds on round trip time (RTT) delay between a pair of hosts, A and B, lie within the interval, $[r_1, r_2]$, and that samples are drawn from a distribution that is independent of time (i.e., the correlation between successive sample values is not a function of the time that elapses between the samples). A protocol is designed with the assumption that the estimate for the round trip time does not vary more rapidly than 1 sec/sec (i.e., over any T units in time, the RTT estimate should not change more than T). Assume the protocol uses an exponentially decaying average with parameter α that indicates the weight of the new sample (i.e., $A((n+1)T) = (1-\alpha)A(nT) + \alpha S_{(n+1)T}$ where S_t is the sample taken at time t, and $A(t)$ is the average computed at time t.) Note that the average varies less with time as the weight, α, is reduced, or as the sampling period, T, is increased. To estimate the round trip time and meet the requirements of the protocol:

   (a) given that the protocol samples the RTT every T seconds, what is the maximum value for α such that the average meets protocol requirements? Explain.

   Answer: To solve this problem, we need to consider the worst-case scenario, i.e., when the average would change most drastically. This happens either when the average is as high as possible (i.e., equal to r_2) and suddenly the smallest sample arrives (i.e., r_1), or when the average is as small as possible (i.e., r_1), and suddenly the largest sample arrives (i.e., r_2.) In the first case, $A(t+T) = (1-\alpha)r_2 + \alpha r_1$, making $A(t) - A(t+T) = \alpha(r_2 - r_1)$. The protocol requires that $\alpha(r_2 - r_1) \leq T$, hence, $\alpha \leq T/(r_2 - r_1)$. Note the same bound holds for the case where the average is r_1 and the next sample is r_2.

   (b) what is the minimum sampling period as a function of α so that the average meets protocol requirements? Explain.

   Answer: We simply solve for T instead of α above, giving $T \geq \alpha(r_2 - r_1)$.

   (c) Describe the difference in how the exponentially weighted average changes with time depending on whether α is varied versus if the sampling period, T, is varied.

   Answer: If α is varied, then by making α smaller, the estimated RTT changes are "smoother", i.e., the sudden jumps in the RTT estimate decrease with decreasing α. By increasing T, the time between the change in estimates increases, and the jumps do not get any smoother, i.e., the change in the estimate between two samples is merely a function of α.

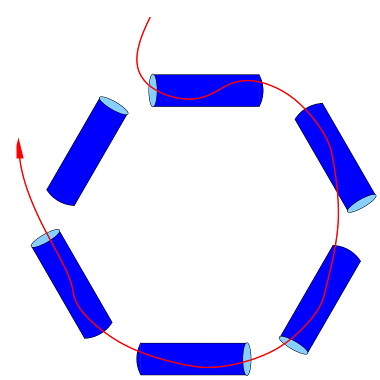


Figure 1: A 6 hop network with one connection traversing 5 of 6links

4. In class (Lecture 4 notes, slide 36), we showed that when flows traverse two links, a situation can arise where increasing throughput rates can in fact lead to a drop in goodput rates as a result of congestion. Set up (but do not solve) the system of equations for flows that traverse n hops each link is the ith hop for one flow for all $0 < i \leq n$. For instance, in Figure 1, $n = 5$ - only one flow is shown, there must be 5 other flows [6 flows in total] , 1 starting at each link). Assume every link L has capacity C_L and let D_L be the rate at which packets (or more appropriately, bits or fluid) is dropped on every link L (the drop rate is the same due to symmetry of the network in the example.

   Helpful hint: The easiest way to proceed is to consider what happens at a single link. Let λ_i equal the rate at which flow enters the link i hops from its source, and let λ_n equal the rate at which a flow exits the link after its nth hop (in the example on p. 36, λ_1 would be relabelled as λ_0, λ_p would be relabeled as λ_1, and λ_x would be relabeled as λ_2). Certain equations will have summations: be specific (and careful) about the range that the summation iterates over.

   Answer: First, if $\lambda_0 \leq C_L/n$, then each flow receives at a rate of λ_0, since no loss occurs.

   For $\lambda_0 > C_L/n$:

   - $\sum_{i=1}^{n} \lambda_i = C_L$
   - $D_L = 1 - C_L / \sum_{i=0}^{n-1} \lambda_i$
   - $\lambda_{i+1} = \lambda_i(1 - D_L) = \lambda_0(1 - D_L)^{i+1}$

5. A sender uses a leaky-bucket mechanism with rate λ and bucket size W to transmit packets into the network. The sender also maintains a window of size W so that the number of sequence numbers in transit at any time is at most W (i.e., more than W packets can appear in the network at a time but then there must be copies of the same packet). Let $D(\lambda)$ be the average delay between the time a packet is transmitted and the time the acknowledgment of its receipt reaches the sender when the sender is transmitting at rate λ. Let p be the probability of packet loss, and assume that if a packet is lost, the sender does not find out about the lost packet for this same amount of time (i.e., it cannot retransmit until time $D(\lambda)$ after the previous transmission of the packet.

   (a) Assume $D(\lambda) = A\lambda + B$ and $p = 0$. What is the maximum rate at which the sender can transmit?

   Answer: By Little's Law, $N = \lambda D(\lambda) = A\lambda^2 + B\lambda$, and because of the window restriction, $N \leq W$. Thus, λ can be no larger than the value where $W = A\lambda^2 + B\lambda$, so $\lambda = (\sqrt{B^2 + 4AW} - B)/2$.

   (b) Suppose $p > 0$ and $D(\lambda) = A'\lambda + B$ and a selective repeat protocol is used. Give an upper bound on the throughput and goodput. Why is this just an upper bound?

   Answer: The throughput is highest when W packets are in transit, but there may be occasions when fewer packets are in transit. For instance, consider the case where a single packet is lost several times. All other packets in the window will eventually be received, and there will be a period of time where this is the only packet in transit. Thus, the throughput is bounded above by assuming that a window W of packets is always in transit: $\lambda \leq (\sqrt{B^2 + 4A'W} - B)/2$. We expect each packet to be transmitted an average of $1/(1-p)$ times, so the upper bound on the goodput, which is the throughput divided by the expected number of times a packet must be transmitted, is $(1-p)\sqrt{B^2 + 4A'W} - B)/2$.

   (c) Suppose the protocol used is Go-Back-N. Does the throughput upper bound still hold? If so, is it a tighter or looser upper bound? What about for the goodput? Explain all answers.

   Answer: The throughput of Go-Back-N will be higher since received packets are retransmitted unnecessarily in Go-Back-N as a result of the acknowledgment system. Hence, the window is more often full, making the upper bound tighter. For the goodput, the transmission of already received packets does not increase the goodput and the window moves forward at the same rate as it does for selective-repeat. Hence, under the described model, the goodput for Go-Back-N should be the same as the goodput for selective repeat, and therefore the upper bound is neither tighter nor looser.

6. Consider an M/M/1/K queue into which packets arrive at rate λ and are serviced at rate μ (assume for simplicity that $\mu \neq \lambda$ and a queue size of K. Consider a queueing discipline in which packets arrive marked as priority with probability p or else they arrive as non-priority. Furthermore, assume that when the processor selects its next packet, it always selects a priority packet if one is available, and that the system is pre-emptive (i.e. it postpones processing of a non-priority packet as soon as there is a priority packet in the system). Furthermore, if the queue is full when a priority packet arrives, then a non-priority packet is discarded to make room for the priority packet.

   (a) What is the loss rate of the system? (Hint: consider the liklihood that the queue is full and apply PASTA)

   Answer: The probability that a packet (whose arrival time is described by a Poisson process) is lost equals the probability that when that packet arrives, the queue is full. By PASTA, that probability is simply the probability that the queue is full, which equals $\rho^K(1-\rho)/(1-\rho^{K+1})$, where $\rho = \lambda/\mu$.

   (b) What is the expected delay experienced by a priority packet?

   Answer: Non-priority packets do not effect the processing of priority packets - so from the perspective of a priority packet, non-priority packets don't even exist in the system. Hence, the delay equals that of a system in which packets arrive at rate $p\lambda$ and are processed at rate μ. This is $1/(\mu - p\lambda)(1-\rho^{K+1})$.

   (c) Assume K is infinite and that $\mu > \lambda$. What is the expected delay experienced by a non-priority packet? (HINT: use the expected value of an "average" packet's delay and the expected value of a high priority packet's delay)

   Answer: Let D be the delay of an "average" packet, let D_H be the delay of a high priority packet, and D_L be the delay of a low priority packet. Let P be a random variable that equals 1 if a packet is high priority and 0 if it is low priority. Then

$$E[D] \quad = \quad \Pr(P = 1)E[D_H] + \Pr(P = 0)E[D_L]$$

   (Note that we used the fact that the queue was infinite size. This way, every packet that enters the system passes through the queue - i.e., we don't need to worry about the distribution of dropped packets)
   We know that $E[D] = 1/(\mu - \lambda)$ and that $E[D_H] = 1/(\mu - p\lambda), \Pr(P = 1) = p$, and $\Pr(P = 0) = 1 - p$. Solving for $E[D_L]$, we get

$$E[D_L] \quad = \quad \frac{\mu}{(\mu - \lambda)(\mu - p\lambda)}$$

   Note that $\mu/\mu - p\lambda > 1$, hence the average delay imposed on a low priority packet is larger than that of the "average" packet through the system, which makes sense since on average high priority packets should go through the system faster than the average low priority packet.

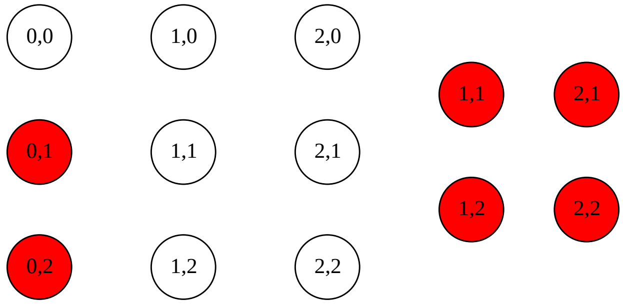
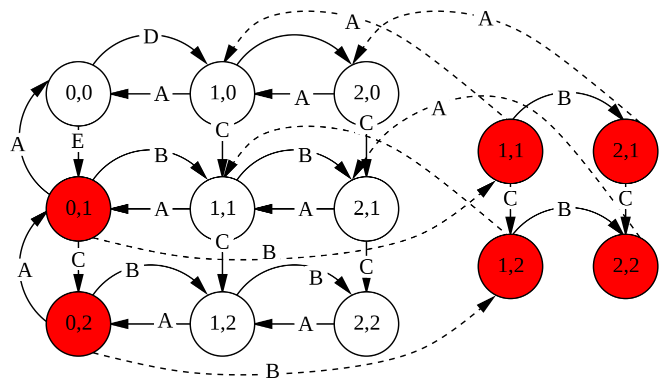


Figure 2: A Markov model for a non-preemptive M/M/1/K queue

(d) Assume now that the queue is non-preemptive. Fill in the transitions and their probabilities for Figure 2 (assume that $K > 2$, but that not all states are shown in the Figure). Shaded states are those in which a low-priority packet is currently being serviced.
Answer:

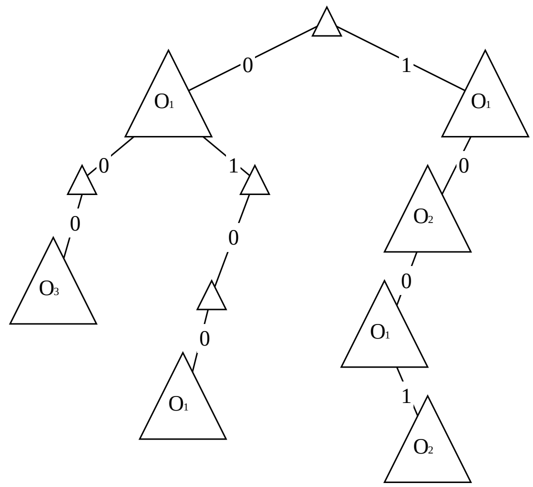


In this figure, $A = \mu/(\mu + \lambda), B = p\lambda/(\mu + \lambda), C = (1 - p)\lambda/(\mu + \lambda), D = p, E = 1 - p$.

7. Draw the trie structure that maps addresses to the interface corresponding to the longest matching prefix in the table below:

prefix	interface
0	1
1	1
10	2
000	3
100	1
1001	2
0100	1

Answer:



8. In this problem, you will build various components of the trie-maintenance algorithm:

   (a) Sometimes there are redundant entries in a lookup table. For instance, consider the following lookup table:

prefix	interface
0	1
1	1
10	1
000	3
0000	3
0001	3
0100	1

   Entries 0000 and 0001 are unnecessary since the entry 000 covers these two cases. It is therefore a waste of memory for the lookup table to contain entries 0000 and 0001. Devise an algorithm called algorithm_clean() that, given a trie, filters out the unncessessary entries (e.g., in this example, 0000 and 0001 would be removed.) Hint: construct a function that can be called by itself recursively.

   Answer: An entry at any point in the trie can be removed so long as the closest upstream point that is marked is marked by the same interface. Assume that Node is a data structure that maintains information about trie nodes, where Node.interface gives the interface that the node maps to, and equals NO_INTERFACE if it doesn't map to an interface. Let Node.left_child point either to the left child (i.e., child headed to when the next bit in the prefix is a 0), or to NULL if there is no left child, and let Node.right_child point either to the right child (the next bit in the prefix is a 1) or to NULL if there is no right child.

   The code shown in Figure 3 does the trick. Let init_node be the initial node in the trie. Then start the process off with traverse_trie(init_node, NO_INTERFACE);

```
traverse_trie(Node curnode, int cur_interface){
   if (curnode.interface != NO_INTERFACE){

   /* current node is currently marked with some interface */
      if (curnode.interface == cur_interface){
         /* matches upstream interface, so no need to have the trie
            mark the interface */
         curnode.interface = NO_INTERFACE;
      }

      /* otherwise curnode indicates a new interface - this is the
         interface that is now the default for prefixes from here
         until the interface changes */
      else cur_interface = curnode.interface;
   }

   /* Recurse down left and right branches (if they exist) */
   if (curnode.left_child != NULL){
      traverse_trie(curnode.left_child, cur_interface);
   }
   if (curnode.right_child != NULL){
      traverse_trie(curnode.right_child, cur_interface);
   }
}
```

Figure 3: Pseudocode to "clean" the trie

prefix	interface
0	1
1	1
10	1
0000	3
00010	3
00011	3
0100	1

(b) Consider the prefix lookup table above. Prefixes 0000, 00010, and 00011 can safely be replaced with the single prefix 000. Briefly describe how you would modify your algorithm to perform such replacements.

Answer: The algorithm would recurse down each branch and return the set of interfaces down that branch. If both branches returned a single interface and the interface was the same on both branches, then the branches can be pruned and that one interface can be assigned to the current point on the trie.