CS 352
Homework #2 Solutions

1. **Give two examples of non-adaptive routing algorithms and three examples of adaptive routing algorithms.**

   Non-adaptive routing algorithms are algorithms that don't update routing table entries based on changes in network topology or network traffic. Examples of non-adaptive routing algorithms include: flooding, simple shortest path routing.

   Adaptive routing algorithms respond to changes in network topology, link state and/or network traffic. Examples of adaptive routing algorithms include: distance vector, link state routing, hot potato routing, backward learning.

2. **Name one way in which distance vector routing is better than link state routing, and name one in which link state routing is better than distance vector routing.**

   Advantage of distance vector routing: Unlike link state routing, which must flood link state update packets to all other routers in the network, distance vectors are only exchanged between adjacent routers. This limits the number of routing packets that are transmitted in the network, thereby saving bandwidth for other data packets in the network.

   Advantage of link state routing: Because they rely on the flooding of link state update packets, link state routing algorithms propagate routing information to routers more rapidly than distance vector algorithms. Furthermore, link state routing doesn't suffer from the count-to-infinity problem, which plagues the distance vector algorithm.

3. **If delays are recorded as 8-bit numbers in a 50-router network, and delay vectors are exchanged twice per second, how much bandwidth per (full-duplex) link is chewed up by the distance vector routing algorithm? Assume that each router is connected to 3 other routers.**

   There is actually some extraneous information in this problem that you didn't need. You didn't need to know that each router is connected to 3 other routers in order to solve this problem.

   To solve this problem, you need to understand how the distance vector algorithm works. First, you must know which routers a given router sends distance vector packets to. In a distance vector algorithm, a router exchanges distance vector packets only with its adjacent routers. So, for the link between two routers A and B, there is a distance vector sent from A to B and a distance vector sent from B to A. Distance vector packets from other routers are *never* sent on this link.

   Second, you must know what and how much information is contained in each distance vector packet. Each distance vector packet contains a single distance metric for every router in the network. In this problem, we are told to use 8-bit delays as distance metrics. Since there are 50 routers, each distance vector carries 50 8-bit distance values. The size of a distance vector packet is therefore equal to $50 \times 8 = 400$ bits.

   Finally, you must know how often these distance vector packets are exchanged. The problem tells us that routers exchange 2 distance vector packets every second.

   Therefore, on a single link, we exchange two 400 bit packets every second in each direction. Thus, we are sending 800 bits per second in each direction. If we consider the total bandwidth used in both directions, we are using a combined total of twice this amount of bandwidth to send distance vector packets. So the answer is 1600 bits per second.

4. **For hierarchical routing with 4800 routers, what region and cluster sizes should be chosen to minimize the size of the routing table for a three-layer hierarchy? (A region is a group of hosts. A cluster is a group of regions.)**

The goal of this problem is to set the region size, cluster size, and number of clusters such that we end up with the smallest possible routing tables when there are a total of 4800 routers.

First let's establish some notation. Let $\mathcal{N}$ be the number of routers per region, let $\mathcal{R}$ be the number of regions per cluster, and let $\mathcal{C}$ be the total number of clusters. A router must have a routing table entry for every other router in its region (including itself). It must also have an entry for every other region (not including its own region). And it must have an entry for every other cluster (not including its own cluster). Thus, each router contains a total of $\mathcal{N} + (\mathcal{R}-1) + (\mathcal{C}-1)$ routing table entries. We also know that the total number of routers is 4800, so the product of $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{C}$ must be greater than or equal to 4800.

Now let's consider a thinking experiment. The reason we use hierarchical routing is to reduce the size of routing tables in the network. So let's contrast hierarchical routing with normal flat (non-hierarchical) routing, which we would expect to produce big routing tables. Flat routing is the same as having 1 cluster ($\mathcal{C}$=1), 1 region per cluster ($\mathcal{R}$=1), and 4800 routers per region ($\mathcal{N}$=4800). How big will each router's routing tables be in this case? The answer is 4800 entries. This is because every router needs to have an entry for every other router in its region. Flat routing produces the worst answer, so we should be able to do better than this with a hierarchical routing solution.

The thinking experiment should have given you some intuition into what the best possible choices of $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{C}$ should have been. In the thinking experiment, the worst solution was obtained by choosing values of $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{C}$ that were as different, or spread out, as possible. This suggests that a better, maybe even the best, choice of values is when the three values are very similar – maybe even equal. So let's consider the case where $\mathcal{N} = \mathcal{R} = \mathcal{C}$. Since we know that the product $\mathcal{NRC}$ must be greater than or equal to 4800, then we can calculate the values of $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{C}$ as follows:

$$\begin{aligned} \mathcal{NRC} &\geq 4800 \\ \mathcal{N}^3 &\geq 4800 \\ \mathcal{N} &\geq 16.85 \quad (\text{similarly } \mathcal{R} \geq 16.85 \text{ and } \mathcal{C} \geq 16.85) \end{aligned}$$

The values of $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{C}$ must be integers (since it is impossible to have a fraction of a routing table entry), so the smallest integer greater than 16.85 is 17. The smallest routing tables are in fact achieved when $\mathcal{N}$ = 17, $\mathcal{R}$ = 17, and $\mathcal{C}$ = 17. This corresponds to routing tables with 49 entries each. 49 is obviously much smaller than 4800, so you should be able to see the benefits of using hierarchical routing as opposed to flat routing.

5. **Name three causes of network deadlock, and come up with a mechanism or algorithm to prevent or alleviate each of them.**

   The three main causes of network deadlock are: direct store and forward lockup, indirect store and forward lockup, and reassembly lockup. (See the lecture notes for descriptions of each.)

   Preventing or alleviating direct store and forward lockup: There are dozens of ways to prevent or alleviate direct store and forward lockup, but I will only list a couple possible answers. One way to stop this type of lockup is to simply eliminate flow control on the link between the adjacent routers. This way, a router does not need to await permission to send a packet to the next downstream router. If a router buffer is full when the packet arrives, then the packet is simply discarded. This is a form of load shedding. Another possible way to stop direct store and forward lockup is to add a timer to the router. Each time the router transmits a packet, it initiates a timer. If the timer expires without sending any additional packets, then the router assumes deadlock is occurring and discards all the packets in its buffer.

   Preventing or alleviating indirect store and forward lockup: The same answers given for direct store and forward lockup apply in the indirect case as well.

   Preventing reassembly lockup: Reassembly lockup can also be prevented or alleviated in several ways. The simplest way to prevent it from occurring is simply to prohibit packet fragmentation. Another way to stop reassembly lockup is to introduce timers for each packet. If after a given amount of time all the segments of a packet have not arrived at the router, then all of the packet's buffered segments are simply discarded.

6. **Does isarithmic flow control completely eradicate congestion from a network? If so, explain how. If not, give an example of a case in which it doesn't.**

   Isarithmic flow control does not necessarily eradicate congestion from a network. If the network ever gets into a state where all or most of the permits are distributed amongst a relatively small number of routers, then congestion (i.e., too many packets) can occur within that small region of routers. Limiting the total number of permits any one router may hold can help alleviate this problem, but it doesn't totally solve it unless the permit limit is set very low, in which case isarithmic flow control wouldn't be very useful anyway.

7. **A computer on a 6-Mbps network is regulated by a token bucket. The token bucket is filled at a rate of 1 Mbps. It is initially filled to capacity with 8 megabits. How long can the computer transmit at the full 6 Mbps?**

   Let us suppose that the computer transmits at 6 Mbps for S seconds. After S seconds, the computer transmits at 1 Mbps because the token bucket has been totally drained of token bits. So, how do we solve for S?

   To solve this problem, we must first realize that the total number of bits transmitted by the computer into the network during the time period S is equal to the total number of token bits consumed by the computer in the same period of time. The total number of bits transmitted by the computer in the time period S is equal to 6 Mbits/sec $\times S$. The total numbers of token bits consumed by the computer in the time period S is equal to the initial token bucket occupancy (8 Mbits) plus the number of token bits generated during the time period S (1 Mbits/sec $\times S$). Hence,

$$6 \text{ Mbits/sec} \times S = 8 \text{ Mbits} + (1 \text{ Mbits/sec} \times S)$$

   Solving for S, we obtain $S = 1.6$ seconds.

8. Most connection establishment protocols rely on a three-way handshake. Why is a three-way handshake protocol better than a two-way handshake?

A three-way handshake is better than a two-way handshake because it allows the communicating parties to detect and recover from duplicate connection establishment packets.

9. If flow control and error control are performed at the data link layer, then why is it also necessary to perform flow and error control at the transport layer?

The transport layer is the first end-to-end layer, whereas the data link layer only connects two adjacent nodes. Providing flow and error control is sufficient to prevent packets from being lost or corrupted on a single link, but it doesn't prevent packets from being discarded or corrupted at a higher layer (namely the network layer). A good example is network layer congestion. If a router discards a packet because one of its buffers overflowed, it doesn't matter that the packet was transmitted reliably by the data link layer – the packet was still discarded at the network layer. In order to recover this packet, we need an error control mechanism at an even higher layer – namely, the transport layer.

10. A TCP sender transmits a 2K-byte packet to the receiver with a sequence number of 0. The receiver has 16K-bytes of buffer space available just before the packet arrives. Assuming the packet remains in the receiver buffer when the receiver generates an acknowledgement, what will the acknowledgement sequence number be? What will the window size be? If the maximum segment size is 2K-bytes, how many more packets can the sender transmit without first waiting for an acknowledgement?

The TCP acknowledgement sequence number is the sequence number of the byte next expected by the receiver. Since the first data packet from the sender contained bytes with sequences numbers 0 through 2047, the acknowledgement sequence number is number 2048. The window size is 16K-2K (or 14K) because there is one packet in the receiver-side buffer when the acknowledgement is generated. Since the window size is 14K, the sender can send up to 7 2K packets before waiting for another acknowledgement.

11. If the smoothed TCP round trip time (SRTT) is currently 30 milliseconds, and the following acknowledgements come in after 26, 32, and 24 msec, respectively, what is the new SRTT using the Jacobson-Karels algorithm? What is the new timeout value? Use α=0.1, δ=0.25, β=4. Assume an initial deviation of 10 msec.

In the Jacobson-Karels algorithm, we calculate the SRTT, the error, the deviation and the timeout values as follows:

$$\begin{aligned}
\text{Error}_{\text{new}} &= \text{RTT}_{\text{measured}} - \text{SRTT}_{\text{old}} \\
\text{SRTT}_{\text{new}} &= \text{SRTT}_{\text{old}} + (\alpha \times \text{Error}_{\text{new}}) \\
\text{Dev}_{\text{new}} &= \text{Dev}_{\text{old}} + \delta \times (|\text{Error}_{\text{new}}| - \text{Dev}_{\text{old}}) \\
\text{Timeout} &= \text{SRTT}_{\text{new}} + \beta \times \text{Dev}_{\text{new}}
\end{aligned}$$

The following table illustrates how the Jacobson-Karels algorithm modifies the SRTT and Timeout values as new RTT measurements are taken:

$\textbf{RTT}_{\textbf{measured}}$	$\textbf{Error}_{\textbf{new}}$	$\textbf{SRTT}_{\textbf{new}}$	$\textbf{Dev}_{\textbf{new}}$	**Timeout**
N/A	N/A	30	10	70
26	-4	29.6	8.5	63.6
32	2.4	29.84	6.975	57.74
24	-5.84	29.256	6.69125	56.021

(All values are expressed in units of milliseconds.)

12. **Suppose that the TCP congestion window is equal to 8K bytes just before a timeout occurs. How big will the congestion window be if the next sixteen packet transmissions are all successfully acknowledged? Assume that the maximum segment size is 1K, all transmitted packets are 1K in length, and the receiver's advertised window is always equal to 64K bytes.**

    When a packet transmission times out (i.e., no acknowledgement for the packet arrives within a timeout period determined by the Jacobson-Karels algorithm), then we have to initiate the *slow start* phase. This means that we set the threshold to half of the current congestion window or half of the current sliding window, whichever is less. In this case, the congestion window (8K) is less than the sliding window (64K), so the threshold is set to 4K. Since we're using slow start, the congestion window is also reduced to 1 maximum segment size – or 1K.

    Because the congestion window is now 1K, the TCP sender transmits one packet before stopping to wait for an acknowledgement. (We can only send one congestion window's worth of data, or one sliding window's worth of data, whichever is less.) **During the slow start phase, the congestion window increases by one segment size whenever one segment of data is acknowledged.** Thus, when the packet's acknowledgement arrives at the sender, the sender increases its congestion window from 1K to 2K. Because the window size is now 2K, the sender sends 2 TCP packets without waiting for an acknowledgement first. When the two acknowledgements return, the congestion window at the source increases from 2K to 4K. So far the sender has transmitted 3 packets since the timeout has occurred.

    The congestion window has now reached the threshold value (4K), so now we say that the sender is no longer in the slow start phase. Instead, it is in the *congestion avoidance* (or linear increase) phase. **During the congestion avoidance phase, the congestion window only increases by one segment size whenever a full congestion window's worth of data has been acknowledged**. Since the congestion window is currently 4K, the congestion window will not advance to 5K until 4 1K packets have been acknowledged. So, after sending 4 more packets and receiving their acknowledgements, the sender's congestion window increases to 5K. After sending 5 more packets and receiving their acknowledgements, the congestion window increases to 6K.

    Since the timeout, the sender has transmitted 12 packets. In order for the congestion window to increase from 6K to 7K, the sender must receive acknowledgements for an additional 6 packets. However, the problem is only interested in the size of the congestion window after 4 more packets have been transmitted (12+4 = 16). So the answer to this problem is that after sixteen packets have been successfully acknowledged, the size of the congestion window remains at 6K.